Magda's notes about **normalizing flows**

Last update: February 18, 2021

Informal notes for my future self who is likely to forget. I explain the papers the way I understand them, using terminology and logic natural to me. This means I may deviate from the original paper structure, notation, etc. At places, my interpretation my be incorrect due to lack of understanding. I will strive for this not to happen too often but I'm certainly not infallible.

This is a working document, not polished, with possible typos, editing errors, etc.

Contents

1 Probability and measure theory

This section contains the prerequisites of measure-theoric view of probability. It is not a full recount of the theory, it only has bits and pieces as and when needed.

Based mainly on Kyle Siegrist. *Random: Probability, Mathematical Statistics, Stochastic Processes*. https://www.randomservices.org

Notes taken: endJan - earlyFeb 2020

1.1 Measure theory

1.1.1 Measurable space

Definition 1.1 A σ**-algebra** $\mathscr{S}$ of a set S is a *non-empty* collection of subsets of S that is closed under a *countable* number of set operations. That is:

- if $A \in \mathscr{S}$ then $A^c \in \mathscr{S}$
- if $A_i \in \mathscr{S}$ for each i in a countable index set I then $\cup_{i \in I} A_i \in \mathscr{S}$.

Additional properties of σ**-algebra** $\mathscr{S}$

- $S \in \mathscr{S}$ (Proof: For any $A \in \mathscr{S}$, we have $A^c \in \mathscr{S}$ and $S = A \cup A^c \in \mathscr{S}$.)
- $\emptyset \in \mathscr{S}$ (Proof: From above $S \in \mathscr{S}$ and $S^c = \emptyset \in \mathscr{S}$.)
- if $A_i \in \mathscr{S}$ for each i in a countable index set I then $\cap_{i \in I} A_i \in \mathscr{S}$ (Proof: If $A_i \in \mathscr{S}$ then $A_i^c \in \mathscr{S}$ and since $B = \cup_{i \in I} A_i^c \in \mathscr{S}$ then $B^c = \cap_{i \in I} A_i \in \mathscr{S}$.)

Definition 1.2 A **measurable space** is the space $(S, \mathscr{S})$ consisting of a set S, and the σ-algebra $\mathscr{S}$.

Definition 1.3 A **Borel** σ**-algebra** is the algebra $\sigma(\mathscr{S})$ generated by the open sets of the topological space $(S, \mathscr{S})$.

Note: Since closed sets are complements of open sets, the Borel σ-algebra contains the closed sets as well (and is in fact generated by the closed sets).

1.1.2 Measurable function

Definition 1.4 Suppose sets S and T and a function $f : S \to T$. If $A \subseteq T$, the **inverse image (pre-image)** of A under f is the subset of S given by $f^{-1}(A) = \{x \in S : f(x) \in A\}$.

Note: Careful, though the notation is the same, the inverse image does not have to be a function (the inverse function may not exist).

Definition 1.5 Suppose sets S and T and a function $f : S \to T$. If $A \subseteq S$, the **forward (direct) image** of A under f is the subset of T given by $f(A) = \{f(x) \in T : x \in A\}$.

Definition 1.6 A **measurable function** is a function $f : S \to T$ where $(S, \mathscr{S})$ and $(T, \mathscr{T})$ are measurable spaces and $f^{-1}(A) \in \mathscr{S}$ for any $A \in \mathscr{T}$.

Note: a continuous function $f : S \to T$ is measurable.

1.1.3 Measure

Definition 1.7 A **positive measure** on $(S, \mathscr{S})$ is the function $\mu : \mathscr{S} \to [0, \infty]$ such that:

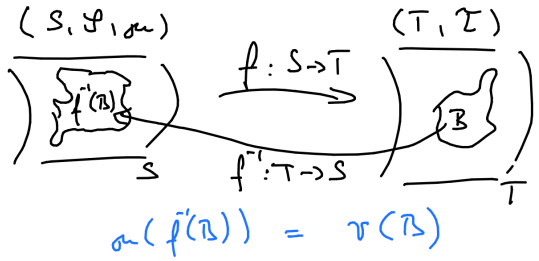
- $\mu(\emptyset) = 0$

- if $\{A_i : i \in I\}$ is a countable, pairwise disjoint collection of sets on $\mathscr{S}$ then $\mu(\cup_{i\in I} A_i) = \sum_{i\in I} \mu(A_i)$ ($\sim$ *countable additivity*).

The triple $(S, \mathscr{S}, \mu)$ is a **measure space**.

Theorem 1.1 (Push-forward measure) Assume a measure space $(S, \mathscr{S}, \mu)$ a measurable space $(T, \mathscr{T})$ and a measurable function $f : S \to T$. Then ν defined as below is a positive measure on $(T, \mathscr{T})$

$$\nu(B) = \mu(f^{-1}(B)), \quad B \in \mathscr{T}$$



Definition 1.8 A **null set** of a measure space $(S, \mathscr{S}, \mu)$ is the set $A \in \mathscr{S}$ such that $\mu(A) = 0$.

Definition 1.9 Consider a statement with $x \in S$ as a free variable.

- The statement **holds on** A if it is true for every $x \in A$
- The statement **holds almost everywhere on** A (with respect to μ) if there exists $B \in \mathscr{S}$ with $B \subseteq A$ such that the statement holds on B and $\mu(A \backslash B) = 0$ (*null set*).

Definition 1.10 Sets $A, B \in \mathscr{S}$ are **equivalent** ($A \equiv B$) if $\mu(A \backslash B) + \mu(B \backslash A) = 0$ (the set of elements in which these two differ has measure zero).

Two measurable functions $f, g : S \to T$ are equivalent if $\mu\{x \in S : f(x) \neq g(x)\} = 0$ (the functions are only different over a set of x with measure zero).

Definition 1.11 Suppose μ and ν are measures on $(S, \mathscr{S})$

a. ν is **absolutely continuous** with respect to μ if every null set of μ is also a null set of ν. We write $\nu \ll \mu$.
b. ν and μ are **mutually singular** if there exists $A \in \mathscr{S}$ such that A is null for μ and A^c is null for ν. We write $\mu \perp \nu$.
c. ν and μ are **equivalent** if $\mu \ll \nu$ and $\nu \ll \mu$. We write $\mu \equiv \nu$.

1.1.4 Some standard measures

Borel measure

Definition 1.12 A topological space $(S, \mathscr{T})$ with $\mathscr{S} = \sigma(\mathscr{T})$ a Borel σ-algebra has a positive measure μ on $(S, \mathscr{S})$ called the **Borel measure**. The triplet $(S, \mathscr{S}, \mu)$ is a **Borel measure space**.

Lebesgue measure

Definition 1.13 (Lebesgue measure) For the standard *Euclidean space* $(\mathbb{R}, \mathscr{R})$, $\mathscr{R}$ is the Borel σ-algebra generated by the standard Euclidean topology (collection of open or closed intervals $\mathscr{I}$ on $\mathbb{R}$).

For $a \leq b \in \mathbb{R}$ all of the intervals $(a,b), (a,b], [a,b), [a,b]$ have the same **length** $b - a$. Intervals $(a,\infty), [a,\infty), (-\infty,a), (-\infty,a]$ have all **length** ∞ and so does $\mathbb{R}$ itself.

There exists a unique measure λ on $\mathscr{R}$ such that $\lambda(I) = length(I)$ for all $I \in \mathscr{I}$. This is the **Lebesgue measure** on $(\mathbb{R}, \mathscr{R})$.

Definition 1.14 (Lebesgue measure) In the *n-dimensional Euclidean space* $(\mathbb{R}^n, \mathscr{R}_n)$, the σ-algebra $\mathscr{R}_n$ is the n-fold power of $\mathscr{R}$, the Borel σ-algebra on $\mathbb{R}$. That is $\mathscr{R}_n = \mathscr{R} \otimes \mathscr{R} \otimes \ldots \otimes \mathscr{R}$. It is also the sigma algebra generated by the products of intervals $\mathscr{R}_n = \sigma\{I_1 \times I_2 \times \ldots \times I_n : I_j \in \mathscr{I}, j \in \{1, 2, \ldots, n\}\}$.

For $n \in \mathbb{N}_+$ the n-fold power of λ denoted λ_n is the **Lebesgue measure** on $(\mathbb{R}^n, \mathscr{R}_n)$. In particular $\lambda(A_1 \times A_2 \times \ldots \times A_n) = \lambda(A_1)\lambda(A_2)\cdots\lambda(A_n)$; $A_1, \ldots, A_n \in \mathscr{R}$ so that λ_n is the *n-dimensional volume* of $A \in \mathscr{R}_n$.

Note: important properties of Lebesgue measure:

Translation invariance If $A \subset \mathbb{R}^n$ and $h \in \mathbb{R}^n$ then $A + h \subseteq \mathbb{R}^n$ and $\lambda_n(A) = \lambda_n(A + h)$

Linear transformation For linear map $T : \mathbb{R}^n \to \mathbb{R}^n$ and $A \subseteq \mathbb{R}^n$ we denote by $TA = \{Tx \in \mathbb{R}^n : x \in A\}$. Then $\lambda_n(TA) = |\det T|\, \lambda_n(A)$ (by the usual interpretation of determinant as measuring the scaling of the linear transformation).

Nonlinear differentiable transformation For differentiable map $f : \mathbb{R}^n \to \mathbb{R}^n$ and $A \subseteq \mathbb{R}^n$ we denote by $f(A) = \{f(x) \in \mathbb{R}^n : x \in A\}$. Then $\lambda_n(f(A)) = |\det J_f|\, \lambda_n(A)$ where $J_f = df_i/dx_j$ is the Jacobian matrix. (This is probably not quite precise - Jacobian evaluated at which point as a local linear approximation?)

Lebesgue-Stieltjes measure

Definition 1.15 A function $F : \mathbb{R} \to \mathbb{R}$ that satisfies

- F is increasing: if $x \leq y$ then $F(x) \leq F(y)$
- F is continuous from the right: $\lim_{t \downarrow x} F(t) = F(x)$ for all $x \in \mathbb{R}$

is a **distribution function** on $\mathbb{R}$.

Definition 1.16 If F is a distribution function on $\mathbb{R}$ then there exists a unique measure μ on $\mathscr{R}$ that satisfies $\mu(a, b] = F(b) - F(a)$, $-\infty \leq a \leq b \leq \infty$. It is called the **Lebesgue-Stieltjes measure**.

Counting measure

Definition 1.17 Assume a finite set S with a power set (set of all subsets) $\mathscr{P}(S)$. For $A \subseteq S$, the cardinality $\#(A)$ is the number of elements in A. The function $\#$ on $\mathscr{P}(S)$ is called the **counting measure**. (It is the usual measure on discrete spaces.)

Probability measure

Definition 1.18 Suppose a measurable space $(S, \mathscr{S})$. A **probability measure** $\mathbb{P}$ is a *positive measure* on $(S, \mathscr{S})$ such that $\mathbb{P}(S) = 1$.

1.2 Probability space

Definition 1.19 A **probability space** $(S, \mathscr{S}, \mathbb{P})$ consists of a set of outcomes of a random experiment S, the σ–algebra of events $\mathscr{S}$, and the probability measure $\mathbb{P}$ on the sample space $(S, \mathscr{S})$.

Note: $(\Omega, \mathscr{F}, \mathbb{P})$ is often used as special notation for probability spaces

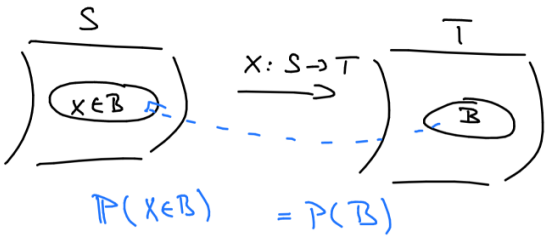
Note: Any finite positive measure μ on the sample space $(S, \mathscr{S})$ can be re-scaled into a probability measure as $\mathbb{P}(A) = \mu(A)/\mu(S)$, $A \in \mathscr{S}$.

Note: Probability measure and **probability distribution** are equivalent terms.

Definition 1.20 (random variable) Suppose that $(S, \mathscr{S}, \mathbb{P})$ is a probability space and $(T, \mathscr{T})$ is another measurable space. A **random variable** X with values in T is a *measurable* function from S to T $(X : S \to T)$.

- the probability measure of X (the *probability distribution* of X) is the mapping $P : T \to [0,1]$ given by $P : B \to \mathbb{P}(X \in B)$ (push-forward measure).
- the collection of events $\{X \in B : B \in \mathcal{T}\}$ is a sub σ-algebra of $\mathcal{S}$ and it is the σ-algebra generated by X denoted by $\sigma(X)$

Note: measurability of X ensure that the inverse image of B under X (that is $\{X \in B\} = \{s \in S : X(s) \in B\}$) is in $\mathcal{S}$ (and hence is a valid event in S) for all $B \in \mathcal{T}$.



Definition 1.21 Suppose that $(S, \mathcal{S}, \mathbb{P})$ is a probability space. Define the following collections of events

- $\mathcal{N} = \{A \in \mathcal{S} : \mathbb{P}(A) = 0\}$, the collection of **null** events
- $\mathcal{M} = \{A \in \mathcal{S} : \mathbb{P}(A) = 1\}$, the collection of **almost sure** events
- $\mathcal{D} = \mathcal{N} \bigcup \mathcal{M} = \{A \in \mathcal{S} : \mathbb{P}(A) = 0 or \mathbb{P}(A) = 1\}$, the collection of **essentially deterministic** events

The collection of essentially deterministic events $\mathcal{D}$ is a sub σ-algebra of $\mathcal{S}$.

Definition 1.22 Random variables X and Y taking values in in T are **equivalent** ($X \equiv Y$) if $\mathbb{P}(X = Y) = 1$. Then

- $\{X \in B\} \equiv \{Y \in B\}$ for every $B \in \mathcal{T}$
- X and Y have the same probability distribution (measure) on $(T, \mathcal{T})$

Definition 1.23 Suppose X is a random variable with values in $\mathbb{R}$. The **probability (cumulative) distribution function** of X is the function $F : \mathbb{R} \to [0,1]$ defined by $F(x) = \mathbb{P}(X \le x)$, $x \in \mathbb{R}$.

The probability distribution function has the following properties

- F is increasing: if $x \le y$ then $F(x) \le F(y)$
- F is continuous from the right: $F(x^+) = F(x)$
- F has limits from the left: $F(x^-) = \mathbb{P}(X < x)$
- $F(-\infty) = 0$
- $F(\infty) = 1$
- If X has continuous distribution on $\mathbb{R}$ then F is continuous.

1.3 Integral with respect to a measure

We denote the integral of a measurable function $f : S \to \mathbb{R}$ with respect to a measure μ as either of the three (usually the first two)

$$\int_S f \, \mathrm{d}\mu, \quad \int_S f(x) \, \mathrm{d}\mu(x), \quad \int_S f(s) \, \mu(dx), \tag{1.1}$$

Definition 1.24 (Integral) Suppose a measure space $(S, \mathcal{S}, \mu)$ with set S, σ-algebra $\mathcal{S}$ and positive measure μ on $\mathcal{S}$

a. For a simple function $f = \sum_{i \in I} a_i \mathbf{1}_{A_i}$ where I is a finite index set and $a_i \in [0, \infty)$ and A_i measurable partition of S then

$$\int_S f \, \mathrm{d}\mu = \sum_{i \in I} a_i \mu(A_i) \tag{1.2}$$

Note: $\int_S \mathbf{1}_A \, \mathrm{d}\mu = \mu(A)$ and $\int_S 0 \, \mathrm{d}\mu = \int_S \mathbf{1}_\emptyset \, \mathrm{d}\mu = 0$.

b. For a measurable $f : S \to [0, \infty)$

$$\int_S f \, \mathrm{d}\mu = \sup \left\{ \int_S g \, \mathrm{d}\mu : \text{ g is simple and } 0 \le g \le f \right\} \tag{1.3}$$

c. For a measurable $f : S \to \mathbb{R}$ with f^+ and f^- denoting the positive and negative parts of f

$$\int_S f \, \mathrm{d}\mu = \int_S f^+ \, \mathrm{d}\mu - \int_S f^- \, \mathrm{d}\mu \tag{1.4}$$

as long as the right side does not form $\infty - \infty$.

d. For a measurable $f : S \to \mathbb{R}$ and $A \in \mathscr{S}$, we define

$$\int_A f \, \mathrm{d}\mu = \int_S \mathbf{1}_A f \, \mathrm{d}\mu \tag{1.5}$$

assuming that the right side exists.

1.3.1 Integral with respect to some common measures

Integration with respect to counting measure $(S, \mathscr{S}, \#)$ is *discrete* if S is countable, $\mathscr{S}$ is collection of all subsets of S, and # is the counting measure on $\mathscr{S}$. Then all $f : S \to \mathbb{R}$ are measurable and integrals with respect to # are simply sums

$$\int_S f \, \mathrm{d}\# = \sum_{x \in S} f(x) \ . \tag{1.6}$$

Lebesgue integration For the one-dimensional Euclidean space $(\mathbb{R}, \mathscr{R}, \lambda)$ where $\mathscr{R}$ is the usual σ-algebra and λ is the Lebesgue measure the *Lebesgue* integral of a measurable function $f : \mathbb{R} \to \mathbb{R}$ over a set $A \in \mathbb{R}$ is

$$\int_A f \, \mathrm{d}\lambda \tag{1.7}$$

Note: **Riemann** integral is denoted as $\int_a^b f(x) \, \mathrm{d}x$. It can be shown that if function f is Riemann integrable on an interval $[a, b]$ it is also Lebesgue integrable and

$$\int_a^b f(x) \, \mathrm{d}x = \int_{[a,b]} f \, \mathrm{d}\lambda \ . \tag{1.8}$$

But not all Lebesgue integrable functions are Riemann integrable. For example the indicator function $\mathbf{1}_\mathbb{Q}$, where $\mathbb{Q}$ is the set of rational numbers in $\mathbb{R}$

- $\int_\mathbb{R} \mathbf{1}_\mathbb{Q} \, \mathrm{d}\lambda = 0$
- $\int_a^b \mathbf{1}_\mathbb{Q}(x) \, \mathrm{d}x$ does not exist for any $a < b$

Note: $f : [a, b] \to \mathbb{R}$ is Riemann integrable on $[a, b]$ if and only if f is bounded on $[a, b]$ and f is continuous almost everywhere on $[a, b]$.

Lebesgue-Stieltjes integral Consider the measurable space $(\mathbb{R}, \mathscr{R})$ and suppose $F : \mathbb{R} \to \mathbb{R}$ is a general distribution function which can be associated with the Lebesgue-Stieltjes measure $\mu(a,b] = F(b) - F(a)$ for all $a < b \in \mathbb{R}$. The *Lebesgue-Stieltjes* integral $\int_s f \, \mathrm{d}\mu$ is often denoted as $\int_S f \, \mathrm{d}F$ or $\int_S f(x) \, \mathrm{d}F(x)$.

Note: If F satisfies the normalizing conditions

- $F(x) \to 0$ as $x \to -\infty$
- $F(x) \to 1$ as $x \to \infty$

then μ is the *probability measure* (def. 1.18) and F is the *probability distribution function* (def. 1.23).

Integration with respect to probability measure Suppose $(S\mathscr{S}, \mathbb{P})$ is a probability space. A measurable real-valued function X on S is a real-valued random variable. The integral of the function X with respect to the probability measure $\mathbb{P}$ is its **expected value**

$$\int_S X \, \mathrm{d}\mathbb{P} = \mathbb{E}(X) \ . \tag{1.9}$$

1.3.2 Product spaces

Suppose that $(S, \mathscr{S}, \mu)$ and $(T, \mathscr{T}, \nu)$ are σ-finite measure spaces and $(S \times T, \mathscr{S} \otimes \mathscr{T}, \mu \otimes \nu)$ the standard product space.

Theorem 1.2 (Fubini's) For a measurable $f : S \times T \to \mathbb{R}$

$$\int_{S\times T} f(x,y) \, \mathrm{d}(\mu \otimes \nu)(x,y) = \int_S \int_T f(x,y) \, \mathrm{d}\nu(y) \, \mathrm{d}\mu(x) = \int_T \int_S f(x,y) \, \mathrm{d}\mu(x) \, \mathrm{d}\nu(y) \tag{1.10}$$

if the double integral on the left exists.

This also implies that

$$\int_{S\times T} g(x)h(y) \, \mathrm{d}(\mu \otimes \nu)(x,y) = \left(\int_S g(x) \, \mathrm{d}\mu(x) \right) \left(\int_T h(y) \, \mathrm{d}\nu(y) \right) \ . \tag{1.11}$$

1.3.3 Change of variable

Suppose a measure space $(S, \mathscr{S}, \mu)$ with set S, σ-algebra $\mathscr{S}$ and positive measure μ on $\mathscr{S}$, another measurable space $(T, \mathscr{T})$ and a measurable function $u : S \to T$. The *push-forward* measure on $\mathscr{T}$ with respect to u is

$$\nu(B) = \mu(u^{-1}(B)), \ B \in \mathscr{T} \tag{1.12}$$

For measurable $f : T \to \mathbb{R}$ we then have

$$\int_T f \, \mathrm{d}\nu = \int_S (f \circ u) \, \mathrm{d}\mu \tag{1.13}$$

or more explicitely

$$\int_T f(u) \, \mathrm{d}\nu(u) = \int_S f(u(x)) \, \mathrm{d}\mu(x) \ . \tag{1.14}$$

1.3.4 Interchange limit and integral

Theorem 1.3 (Monotone convergence theorem) Suppose $f_n : S \to [0,\infty)$ is measurable for $n \in \mathbb{N}_+$ and that f_n is increasing in n. Then

$$\int_S \lim_{n\to\infty} f_n \, \mathrm{d}\mu = \lim_{n\to\infty} \int_S f_n \, \mathrm{d}\mu \tag{1.15}$$

Theorem 1.4 (Dominated convergence theorem) Suppose $f_n : S \to [0,\infty)$ is measurable for $n \in \mathbb{N}_+$ and $\lim_{n\to\infty} f_n$ exists on S. Suppose also that $|f_n| \le g$ for $n \in \mathbb{N}$ where $g : S \to [0,\infty)$ is integrable. Then

$$\int_S \lim_{n\to\infty} f_n \, \mathrm{d}\mu = \lim_{n\to\infty} \int_S f_n \, \mathrm{d}\mu \tag{1.16}$$

1.4 Density functions

Definition 1.25 Suppose μ is a measures on $(S,\mathscr{S})$ and $f : S \to \mathbb{R}$ is measurable and integrable with respect to μ. The function ν defined as

$$\nu(A) = \int_A f \, \mathrm{d}\mu, \; A \in \mathscr{S} \tag{1.17}$$

is a σ-finite measure on $(S,\mathscr{S})$ that is absolutely continuous with respect to μ. The function f is the **density function** of ν relative to μ.

The most important special cases are

- if f is non-negative then ν is a *positive measure* since $\nu(A) \ge 0$ for $A \in \mathscr{A}$
- if f is integrable then ν is a *finite measure* since $\nu(A) \in \mathbb{R}$ for $A \in \mathscr{A}$
- if f is non-negative and $\int_S f \, \mathrm{d}\mu = 1$ then ν is a *probability measure* since $\nu(A) \ge 0$ for $A \in \mathscr{A}$ and $\nu(S) = 1$.

Theorem 1.5 (uniqueness of density function) Suppose ν is a σ-finite measure on $(S,\mathscr{S})$ and that ν has a density function f with respect to μ. Then $g : S \to \mathbb{R}$ is a density function of ν with respect to μ if and only if $f = g$ almost everywhere on S with respect to μ.

Theorem 1.6 Suppose ν is a σ-finite measure on $(S,\mathscr{S})$

Lebesgue decomposition ν can be uniquely decomposed into $\nu = \nu_c + \nu_s$ where $\nu_c \ll \mu$ and $\nu_s \perp \mu$.

Radon-Nikodym ν_c has a density function with respect to μ.

Theorem 1.7 (Radon-Nikodym) Suppose ν and μ are σ-finite measures on $(S,\mathscr{S})$. If $\nu \ll \mu$ then there exists a measurable function $f : S \to \mathbb{R}$ such that for any $A \in \mathscr{S}$

$$\nu(A) = \int_A f \, \mathrm{d}\mu \left(= \int_A \mathrm{d}\nu \right) . \tag{1.18}$$

The function f is called **Radon-Nikodym derivative** of ν with respect to the measure μ, $f \, \mathrm{d}\mu = \mathrm{d}\nu$ and is often denoted as $\mathrm{d}\nu/\mathrm{d}\mu$. The *Radon-Nikodym derivative* is an alternative name for the *density function* of ν with respect to μ.

Note: Careful, remember that the density function f is unique only up to the μ-null set.

Theorem 1.8 (change of variable) Suppose ν and μ are σ-finite measures on $(S,\mathscr{S})$ with $\nu \ll \mu$ and f a density function of ν with respect to μ. If $g : S \to \mathbb{R}$ whose integral with respect to ν exists then

$$\int_S g \, \mathrm{d}\nu = \int_S gf \, \mathrm{d}\mu \tag{1.19}$$

or simply $\mathrm{d}\nu = f \, \mathrm{d}\mu$.

1.4.1 Some special cases

Discrete measure space $(S, \mathscr{S}, \#)$ is *discrete* if S is *countable* and $\mathscr{S} = \mathscr{P}(S)$ is the power set, that is the collection of all subsets of S, and # is the counting measure on $\mathscr{S}$ (positive and σ-finite) with $\emptyset$ the only null set for #. Assume ν is an absolutely continuous measure relative to μ. By the Radon-Nikodym theorem, ν can be written as

$$\nu(A) = \int_A f \, \mathrm{d}\# = \sum_{x \in A} f(x), \; A \subseteq S \tag{1.20}$$

for a unique $f : S \to \mathbb{R}$.

1.5 Probability density function

Suppose that $(\Omega, \mathscr{F}, \mathbb{P})$ is a probability space, $(S, \mathscr{S})$ is another measurable space, and $X : \Omega \to S$ is a random variable (measurable function). Probability distribution of X is the *push-forward* probability measure P on $(S, \mathscr{S})$ defined as

$$P(A) = \mathbb{P}(X \in A) \;, \tag{1.21}$$

where $\{X \in A\}$ is the *inverse image* of A under X.

Discrete probability space

Definition 1.26 Suppose r.v. $X \in S$ taking values in a discrete measure space $(S, \mathscr{S}, \#)$. Then X has a **discrete probability distribution** (probabiliyty measure) $P \ll \#$ with a density function f defined by $f(x) = P(x) = \mathbb{P}(X = x)$ for $x \in S$ with the following properties

- $f(x) \geq 0, \; x \in S$
- $P(S) = \int_S f \, \mathrm{d}\# = \sum_{x \in S} f(x) = 1$
- $P(A) = \int_A f \, \mathrm{d}\# = \sum_{x \in A} f(x)$ for $A \subseteq S$

Note: Conversely, a non-negative function f on S such that $\sum_{x \in S} f(x) = 1$ is a (discrete) probability density function on S with a probability measure P on S defined as $P(A) = \sum_{x \in A} f(x)$ for $A \subseteq S$.

Note: Discrete probability density function is equivalent to discrete **probability mass function** with total mass 1.

Continuous probability space

Definition 1.27 Suppose r.v. $X \in S$ taking values in a measurable space $(S, \mathscr{S})$ with $S \in \mathbb{R}^n$ of n-dimensional Euclidean space $(\mathbb{R}^n, \mathscr{R}^n, \lambda_n)$ with the standard Lebesgue measure λ_n. Then X has a **continuous probability distribution** (probability measure) P if $P(x) = \mathbb{P}(X = x) = 0$ for all $x \in S$.

The distribution P has a density function f with respect to λ_n if $P \ll \lambda_n$ so that if $\lambda_n(A) = 0$ then $P(A) = \mathbb{P}(X \in A) = 0$ for all $A \in \mathscr{S}$ defined as

$$P(A) = \int_A f \, \mathrm{d}\lambda_n, \; A \in \mathscr{S} \;. \tag{1.22}$$

Note: Continuous distributions spread the probability mass continuously over S while discrete distributions concentrate the probability mass on the points of the discrete set S.

Note: $\int_S f(x) \, \mathrm{d}\lambda_n(x) = P(S) = 1$.

Note: For $S \subset \mathbb{R}^n$ we can always extend the density f to all $\mathbb{R}^n$ by setting $f(x) = 0$ for $x \notin S$.

Expected value

Definition 1.28 Suppose a measurable function $r : S \to \mathbb{R}$ so that $r(X)$ is a real-valued r.v. The integral of $r(X)$ with respect to the probability measure is the *expected value* of $r(X)$. Using the change of variable we have for the **expected value**

$$\mathbb{E}[r(X)] = \int_S r(x)\, \mathrm{d}P(x) = \int_\Omega r(X(\omega))\, \mathrm{d}\mathbb{P}(\omega) \ . \tag{1.23}$$

If further P has a density f with respect to a measure μ on $(S, \mathscr{S})$ (e.g. λ_n or #) then by the change of variable (1.19)

$$\mathbb{E}[r(X)] = \int_S r(x)\, \mathrm{d}P(x) = \int_S r(x) f(x)\, \mathrm{d}\mu(x) \tag{1.24}$$

Furhter, let F_Y denote the probability (cumulative) distribution function (def. 1.23) of $Y = r(X)$. Y has a probability measure P_Y on $\mathbb{R}$ which is also the Lebesgue-Stieltjes measure

$$P_Y(a, b] = \mathbb{P}(a < Y \leq b) = F_Y(b) - F_Y(a);\ a < b \in \mathbb{R} \ . \tag{1.25}$$

Using the change of variable we have for the expected value

$$\mathbb{E}[r(X)] = \int_S r(x)\, \mathrm{d}P(x) = \int_\Omega r(X(\omega))\, \mathrm{d}\mathbb{P}(\omega) = \int_{\mathbb{R}} y\, \mathrm{d}P_Y(y) = \int_{\mathbb{R}} y\, \mathrm{d}F_Y(y) \ , \tag{1.26}$$

where the last equality is just a change of notation.

1.5.1 Relation between density and distribution function

For X with *discrete distribution* on countable set $S \subseteq \mathbb{R}$ and f and F the probability density function and the distribution function respectively (def. 1.23)

- $F(x) = \sum_{t \in S : t \leq x} f(t)$ for $x \in \mathbb{R}$
- $f(x) = F(x) - F(x^-)$ for $x \in S$

For X with *continuous distribution*

- $F(x) = \int_{-\infty}^{x} f(t)\, \mathrm{d}t = \int_{(-\infty, x]} f(t)\, \mathrm{d}\lambda(t) =$ for $x \in \mathbb{R}$
- $f(x) = F'(x) = \frac{\mathrm{d}F}{\mathrm{d}\lambda}(x)$ if f is continuous at x

1.6 Transformation of random variables

Assume

- random experiment with a probability space $(\Omega, \mathscr{F}, \mathbb{P})$
- random variable $X : \Omega \to S$ for the experiment taking values in S
- function $r : S \to T$

Then $Y = r(X)$ is a new random variable taking values in T. Further, recall that the inverse image of $B \in T$ under r is $r^{-1}(B) = \{x \in S : r(x) \in B\}$.

Then $\mathbb{P}(Y \in B) = \mathbb{P}(r(X) \in B) = \mathbb{P}(X \in r^{-1}(B))$ for $B \in T$.

For X with *discrete distribution* on countable set S with probability density function f, $Y = r(X)$ has discrete distribution with density function

$$g(y) = \sum_{x \in r^{-1}(y)} f(x),\ y \in T \tag{1.27}$$

Proof: This follows from the countable additivity $g(y) = \mathbb{P}(Y = y) = \mathbb{P}(X \in r^{-1}(y)) = \sum_{x \in r^{-1}(y)} f(x)$.

For X with *continuous distribution* on a set $S \subset \mathbb{R}^n$ with probability density function f, $Y = r(X)$ has continuous distribution with density function

$$g(y) = \int_{r^{-1}(y)} f(x)\,\mathrm{d}\lambda(x),\ y \in T \tag{1.28}$$

Proof: Same as for discrete.

For X with *continuous distribution* on a set $S \subset \mathbb{R}^n$ with probability density function f and $T \subseteq \mathbb{R}$, $Y = r(X)$ has continuous distribution with (cummulative) distribution function (def .1.23)

$$G(y) = \int_{r^{-1}(-\infty,y]} f(x)\,\mathrm{d}\lambda(x),\ y \in \mathbb{R} \tag{1.29}$$

Proof: $G(y) = \mathbb{P}(Y \le y) = \mathbb{P}[r(X) \in (-\infty, y]] = \mathbb{P}[X \in r^{-1}(-\infty, y]] = \int_{r^{-1}(-\infty,y]} f(x)\,\mathrm{d}x$.

1.6.1 Change of variable formula for densities

Univariate (scalar) random variables Suppose X is a r.v. with *continuous distribution* on an interval $S \subset \mathbb{R}$ with probability density function f and distribution funciton F. Suppose $Y = r(X)$, where $r : S \to T$ (to an interval $T \subseteq \mathbb{R}$) is invertible and both r and r^{-1} are differentiable (*diffeomorphism*) . Denote by g and G the density and distribution functions of Y.

Suppose r is *strictly increasing* on S, for $y \in T$

- $G(y) = F\left(r^{-1}(y)\right)$
- $g(y) = f\left(r^{-1}(y)\right)\frac{d}{dy}r^{-1}(y)$

Proof: $G(y) = \mathbb{P}(Y \le y) = \mathbb{P}(r(X) \le y) = \mathbb{P}(X \le r^{-1}(y)) = F\left(r^{-1}(y)\right)$. From relation between density and distribution function $F' = f$ (section 1.5.1) we have $g(y) = G'(y) = \frac{d}{dy}F\left(r^{-1}(y)\right) = f\left(r^{-1}(y)\right)\frac{d}{dy}r^{-1}(y)$.

Suppose r is *strictly decreasing* on S, The for $y \in T$

- $G(y) = 1 - F\left(r^{-1}(y)\right)$
- $g(y) = -f\left(r^{-1}(y)\right)\frac{d}{dy}r^{-1}(y)$

Proof: $G(y) = \mathbb{P}(Y \le y) = \mathbb{P}(r(X) \le y) = \mathbb{P}(X \ge r^{-1}(y)) = 1 - F\left(r^{-1}(y)\right)$, and $g(y)$ follows as above.

Since for strictly decreasing function $\frac{d}{dy}r^{-1}(y) < 0$ we can merge these together into a single result.

Theorem 1.9 For r *strictly increasing or decreasing* on S, for $y \in T$

$$g(y) = f\left(r^{-1}(y)\right)\left|\frac{d}{dy}r^{-1}(y)\right| \tag{1.30}$$

Alternative proof using Lebesgue integration

By Radon-Nikodym theorem (theorem 1.7)

$$\int_{(-\infty,y]} g(t)\,\mathrm{d}\lambda(t) = G(y) = F\left(r^{-1}(y)\right) = \int_{(-\infty,r^{-1}(y)]} f(s)\,\mathrm{d}\lambda(s) \tag{1.31}$$

By change of variable and properties of Lebesgue measure (transformation)

$$\int_{(-\infty,r^{-1}(y)]} f(s)\,\mathrm{d}\lambda(s) = \int_{(-\infty,y]} f(r^{-1}(t))\,\mathrm{d}\lambda(r^{-1}(t)) = \int_{(-\infty,y]} f(r^{-1}(t))|\det J_{r^{-1}}(t)|\,\mathrm{d}\lambda(t) \tag{1.32}$$

This implies that

$$g(t) = f(r^{-1}(t))|\det J_{r^{-1}}(t)| = f\left(r^{-1}(t)\right)\left|\frac{d}{dt}r^{-1}(t)\right| \tag{1.33}$$

Multivariate (vector) random variables

Theorem 1.10 Suppose $\mathbf{X}$ is a r.v. with *continuous distribution* on $S \subset \mathbb{R}^n$ with probability density function f and distribution funciton F. Suppose $\mathbf{Y} = r(\mathbf{X})$, where $r : S \to T \subseteq \mathbb{R}^n$ is *diffeomorphism* and denote by g and G the density and distribution functions of $\mathbf{Y}$. Then the probability density function g of $\mathbf{Y}$ is given by

$$g(\mathbf{y}) = f(r^{-1}(\mathbf{y}))\left|\det J_{r^{-1}}(\mathbf{y})\right| \ , \tag{1.34}$$

where $J_{r^{-1}}(\mathbf{y}) = \frac{d}{d\mathbf{y}} r^{-1}(\mathbf{y})$ is the Jacobian matrix and the determinant of the Jacobian describes how the n-dimensional volume changes under the transformation r.

Proof with Riemann integration: $\mathbb{P}(\mathbf{Y} \in B) = \int_B g(\mathbf{y})\, \mathrm{d}\mathbf{y}$ and at the same time $\mathbb{P}(\mathbf{Y} \in B) = \mathbb{P}(r(\mathbf{X}) \in B) = \mathbb{P}(\mathbf{X} \in r^{-1}(B)) = \int_{r^{-1}(B)} f(\mathbf{x})\, \mathrm{d}x$.

Using the change of variables $\mathbf{x} = r^{-1}(\mathbf{y})$, and $\mathrm{d}\mathbf{x} = \left|\det J_{r^{-1}}(\mathbf{y})\right| \mathrm{d}\mathbf{y}$

$$\int_{r^{-1}(B)} f(\mathbf{x})\, \mathrm{d}x = \int_B f(r^{-1}(y))\left|\det J_{r^{-1}}(\mathbf{y})\right| \mathrm{d}\mathbf{y} = \int_B g(\mathbf{y})\, \mathrm{d}\mathbf{y} \tag{1.35}$$

and hence the result for $g(\mathbf{y})$ in (1.34).

Proof with Lebesgue integration: follows in analogy from the univariate $\mathbb{R}$ case.

1.6.2 Special transformations

Linear (location-scale) transformation Suppose X is a r.v. in $S \subseteq \mathbb{R}$ with a continuous distribution on S with the density function f. Let $Y = a + bX$, where $a \in \mathbb{R}$, $b \in \mathbb{R}\backslash\{0\}$. Then $Y \in T = \{y = a + bx : x \in S\}$. The probability density function g of Y is given by

$$g(y) = f\left(\frac{y-a}{b}\right)\frac{1}{|b|} \tag{1.36}$$

Proof: $y = a + bx \Rightarrow x = (y-a)/(b)$ and $\mathrm{d}x/\,\mathrm{d}y = 1/b$. Plug these into (1.30).

Suppose $\mathbf{X}$ is a r.v. in $S \subseteq \mathbb{R}^n$ with a continuous distribution on S with the density function f. Let $\mathbf{Y} = \mathbf{a} + \mathbf{B}\mathbf{X}$, where $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{B}$ is invertible matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$. Then $\mathbf{Y} \in T = \{\mathbf{y} = \mathbf{a} + \mathbf{B}\mathbf{x} : \mathbf{x} \in S\}$. The probability density function g of $\mathbf{Y}$ is given by

$$g(\mathbf{y}) = f\left(\mathbf{B}^{-1}(\mathbf{y} - \mathbf{a})\right)\frac{1}{|\det \mathbf{B}|} \tag{1.37}$$

Proof: $\mathbf{y} = \mathbf{a} + \mathbf{B}\mathbf{x} \Rightarrow \mathbf{x} = \mathbf{B}^{-1}(\mathbf{y} - \mathbf{a})/(b)$ and $\det \mathbf{B}^{-1} = 1/(\det \mathbf{B})$. Plug these into (1.34).

Sums and convolutions Suppose X and Y are r.v.s in $R \subseteq \mathbb{R}$ and $S \subseteq \mathbb{R}$ respectively so that $(X, Y) \in R \times S$ with probability density function f. Let $Z = X + Y$, then $Z \in T = \{z = x + y : x \in R, y \in S\}$. For $z \in T$ let $D_z = \{x \in R : z - x \in S\}$.

If (X, Y) has a *discrete* distribution than $Z = X + Y$ has a *discrete* distribution with density function u given by

$$u(z) = \sum_{x \in D_z} f(x, z - x),\ z \in T \tag{1.38}$$

Proof: $\mathbb{P}(Z = z) = \mathbb{P}(X = x, Y = z - x \text{ for some } x \in D_z) = \sum_{x \in D_z} f(x, z - x)$

If (X, Y) has a *continuous* distribution than $Z = X + Y$ has a *continuous* distribution with density function u given by

$$u(z) = \int_{x \in D_z} f(x, z - x)\, \mathrm{d}x,\ z \in T \tag{1.39}$$

Proof: For $A \subseteq T$ let $C = \{(a,b) \in R \times S : a+b \in A\}$. Then $\mathbb{P}(Z \in A) = \mathbb{P}(X+Y \in A) = \int_C f(a,b)\,\mathrm{d}(a,b)$ Use change of variable $x = a, z = a+b \Rightarrow a = x, b = z-x$ with Jacobian determinant $\det \mathrm{d}(a,b)/\mathrm{d}z = 1$. Using the change of variable

$$\mathbb{P}(Z \in A) = \int_C f(a,b)\,\mathrm{d}(a,b) = \int_{D_z \times A} f(x,z-x)\,\mathrm{d}(x,z) = \int_A \int_{D_z} f(x,z-x)\,\mathrm{d}x\,\mathrm{d}z \tag{1.40}$$

Suppose X and Y are **independent** r.v.s with density functions g and h respectively. Then their **sum** $Z = X+Y$ has a density $u = g * h$ given by the **convolution**

For X, Y and Z *discrete*

$$u(z) = (g*h)(z) = \sum_{x \in D_z} g(x)h(z-x),\ z \in T \tag{1.41}$$

For X, Y and Z *continuous*

$$u(z) = (g*h)(z) = \int_{D_z} g(x)h(z-x)\,\mathrm{d}x,\ z \in T \tag{1.42}$$

Proof: Both results follow from the general results by $f(x,y) = g(x)h(y)$.

2 Dinh et al.: RealNVP

Paper: Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density Estimation Using Real NVP". in: *ICLR*. 2017

Notes taken: 6/2/2020

TLDR: Unsupervised generative model relying on the *change of variable formula* for probability densities

$$p_X(\mathbf{x}) = p_Z(g^{-1}(\mathbf{x}))\left|\det J_{g^{-1}}(\mathbf{x})\right| \ , \tag{2.1}$$

where $\mathbf{z} \sim p_Z$ is a latent variable generated from an arbitrary (simple) distribution with density p_Z (e.g. standard Gaussian) and we learn the transformation $\mathbf{x} = g(\mathbf{z})$ to get the data distribution p_X (the random vars $\mathbf{Z}$ and $\mathbf{X}$ need to have the same number of dimensions).
The learned function g has to be sufficiently flexible to enable transforming the simple p_Z to arbitrarily complex p_X. Yet it also has to be invertible to be able to perform both inference and sampling. Furthermore the inverse and the determinant of the Jacobian $J_{g^{-1}}$ should not be too expensive to get to be able to train efficiently.
They propose to construct g as a stacking of **affine coupling layers**

$$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp\Big(s(\mathbf{x}_{1:d})\Big) + t(\mathbf{x}_{1:d}) \ , \end{aligned} \tag{2.2}$$

where s and t are neural networks. Each of these layers has triangular Jacobian with cheap determinant and easy inverse, and the determinant and inverse of the whole stacking is just a product of determinants and inverses of each layer.
The partitions need to be alternated to ensure that all dimensions get updated and can interact. They implement the partitioning by masking.

2.1 Intro

Generative probabilistic modelling for high-dimensional and highly structured data yet still trainable = *real-valued non-volume preserving (real NVP) transformations*. The model enables efficient and exact inference, sampling and density estimation.

VAEs train a generative network and an approximate inference network and optimize a lower bound on the log-likelihood. The approximate inference limits the ability to learn good representations.

Autoregressive models are tractable models of the log-likelihood but the arbitrary ordering of the dimensions in the conditioning chain in the joint distribution may be critical for the performance. Also these are sequential and hence non-parallelizable so computationally not efficient.

GANs don't optimize likelihood but instead use a discriminator network to provide a training signal to the generative network. Measures for diversity of the generated samples are currently intractable.

2.2 Model

Generative network $g : z \to x$ mapping latent variable $z \sim p_Z$ to a sample $x \sim p_X$ (p are the respective densities) can be trained through maximum likelihood, if g is *bijective*, using the **change of**

variable formula (see details and proofs in section 1.3.3)

$$p_X(\mathbf{x}) = p_Z(g^{-1}(\mathbf{x}))\left|\det J_{g^{-1}}(\mathbf{x})\right| \tag{2.3}$$

$$\log p_X(\mathbf{x}) = \log p_Z(g^{-1}(\mathbf{x})) + \log\left|\det J_{g^{-1}}(\mathbf{x})\right| \tag{2.4}$$

Some useful math:

$$\det(AB) = \det(A)\det(B) \qquad \det(A^{-1}) = \Big(\det(A)\Big)^{-1} \tag{2.5}$$

For $f : \mathbb{R} \to \mathbb{R}$ and $b = f(a)$ and $F : \mathbb{R}^n \to \mathbb{R}^n$ and $\mathbf{q} = F(\mathbf{p})$

$$\Big(f^{-1}\Big)'(b) = \frac{1}{f'(a)} = \frac{1}{f'\Big(f^{-1}(b)\Big)} \qquad J_{F^{-1}}(\mathbf{q}) = \Big(J_F(\mathbf{p})\Big)^{-1}, \tag{2.6}$$

where $J_F(\mathbf{p}) = \frac{\partial F}{\partial \mathbf{p}}(\mathbf{p})$ is the Jacobian matrix evaluated at $\mathbf{p}$.

2.2.1 Coupling layers

Computing Jacobian and its determinant is expensive in high dimensions but if Jacobian is upper- or lower- triangular then the determinant is just a product of the diagonal terms.

We can build flexible but tractable bijective function by stacking simple bijections.

They propose to stack **affine coupling layers** operating over the inputs $\mathbf{x} \in \mathbb{R}^D$ and spitting out the outputs $\mathbf{y} \in \mathbb{R}^D$ as follows ($d < D$):

$$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp\Big(s(\mathbf{x}_{1:d})\Big) + t(\mathbf{x}_{1:d}) \ , \end{aligned} \tag{2.7}$$

where s and t are scale and translation functions $\mathbb{R}^d \to \mathbb{R}^{D-d}$.

Jacobian of the coupling layer is triangular

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{I}_d & \mathbf{0} \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \operatorname{diag}\Big(\exp\Big(s(\mathbf{x}_{1:d})\Big)\Big) \end{bmatrix} \tag{2.8}$$

so that the determinant is just

$$\det \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \exp\Big(\sum_j s(\mathbf{x}_{1:d})_j\Big) \ . \tag{2.9}$$

As computing the Jacobian of the coupling layer does not involve computing the Jacobians of s or t, these can be arbitrarily complex and are modelled by neural nets.

Inverse of the coupling layer is easy meaning that the sampling is also computationally efficient

$$\begin{aligned} \mathbf{x}_{1:d} &= \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} &= \Big(\mathbf{y}_{d+1:D} - t(\mathbf{y}_{1:d})\Big) \odot \exp\Big(-s(\mathbf{y}_{1:d})\Big) \ , \end{aligned} \tag{2.10}$$

They propose to implement the **partitioning via binary mask** b

$$\mathbf{y} = \mathbf{b} \odot \mathbf{x} + (1 - \mathbf{b}) \odot \Big(\mathbf{x} \odot \exp\Big(s(b \odot \mathbf{x})\Big) + t(b \odot \mathbf{x})\Big) \tag{2.11}$$

To ensure that all components of the inputs **x** get transformed, they apply the coupling layers in **alternating pattern swapping the fixed and the transformed parts**. The Jacobian of the stacking remains tractable because by the chain rule and the math above

$$\det\left(\frac{\partial(g \circ h)}{\partial x}(\mathbf{x})\right) = \det\left(\frac{\partial g}{\partial y}(\mathbf{y} = h(\mathbf{x}))\right)\det\left(\frac{\partial h}{\partial x}(\mathbf{x})\right) \quad . \tag{2.12}$$

The inverse is also tractable by

$$(g \circ h)^{-1} = h^{-1} \circ g^{-1} \quad . \tag{2.13}$$

2.3 Experiments

They experiment with standard image datasets (CIFAR-10, small Imagenet, etc.). They claim that compared to VAE their generations are sharper and that *as well known* optimizing for likelihood values diversity over sample quality. The bits/dimension metric is somewhat worse than in pixelRNN but they claim this could be improved by larger model.

They also explore how smoothly varying the latent space translate to meaningful changes in the output space which goes beyond just altering the pixel values. This looks interesting but is only coming out from the experiments, not from the theory.

2.3.1 Implementation tricks

They explain a couple of tricks to make the implementation faster and more stable.

- multi-scale architecture with squeezing - a way to alternate the coupling layers
- factoring out half of dimensions at regular intervals - to alleviate the need for propagating all dimensions through all coupling layers
- batch normalization using running averages - robust to training with small minibatches

Check sections 3.6-3.7 of the paper if implementation is of the essence.

2.3.2 Conclusions

Possible future work is extending this to semi-supervised and/or conditionally generative model. In appendix F they also show a bit on conditional generations but it is not very clear how they did this.

3 Dinh et al.: NICE

Paper: Laurent Dinh, David Krueger, et al. "NICE: Non-Linear Independent Components Estimation". In: *ICLR (Workshop)*. 2015

Notes taken: 11/2/2020

TLDR: Learn the transformation $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^D$, $\mathbf{H} = \mathbf{f}(\mathbf{X})$ from data $\mathbf{X}$ to a hidden variable $\mathbf{H}$ with a factorial distribution $p_H(\mathbf{h}) = \prod_i^D p_{H_i}(h_i)$ by maximizing the log likelihood

$$\log p_X(\mathbf{x}) = \sum_{i=1}^{D} \log p_{H_i}(\mathbf{f}_i(\mathbf{x})) + \log \left| \det \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \right| . \tag{3.1}$$

The transformation $\mathbf{f}$ is a stacking of **additive coupling layers** operating over two blocks of the data vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$

$$\mathbf{h}_1 = \mathbf{x}_1 \qquad \mathbf{h}_2 = \mathbf{x}_2 + \mathbf{m}(\mathbf{x}_1) \tag{3.2}$$

which have very simple inverse $\mathbf{f}^{-1}$

$$\mathbf{x}_1 = \mathbf{h}_1 \qquad \mathbf{x}_2 = \mathbf{h}_2 - \mathbf{m}(\mathbf{h}_1) \tag{3.3}$$

to be able to do the ancestral sampling $\mathbf{h} \sim p_H(\mathbf{h})$, $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{h})$ and trivial Jacobian determinant $\det \frac{\partial \mathbf{h}}{\partial \mathbf{x}} = 1$ so that we can easily evaluate and maximize the likelihood. They add element-wise scaling factors S_{ii} to the last transformation in the stack to give more weight to some dimensions of the latent variable and achieve a PCA effect. This makes the objective only slightly more complex as the the $\det J$ is just the product $\det \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \prod_i S_{ii}$.

3.1 Intro

This was written before the realNVP (Dinh, Sohl-Dickstein, and Bengio 2017) with the same 1st author.

How to capture complex data distribution? "Good representation is one in which the data distribution is easy to model".

Find transformation of the data $\mathbf{X}$ to a latent variable $\mathbf{H} = \mathbf{f}(\mathbf{X})$ such that the resulting distribution of the r.v. $\mathbf{H}$ factorizes across the D

$$p_H(\mathbf{h}) = \prod_{i}^{D} p_{H_i}(h_i) \ . \tag{3.4}$$

The transformation $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^D$ should be invertible and differentiable so that by the change of variable we get

$$p_X(\mathbf{x}) = p_H(\mathbf{f}(\mathbf{x})) \left| \det \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \right| . \tag{3.5}$$

The prior distribution p_H is some predefined fixed distribution (e.g. isotropic Gaussian). We want $\mathbf{f}$ with simple inverse $\mathbf{f}^{-1}$ so that we can easily sample $\mathbf{x}$ by ancestral sampling

$$\mathbf{h} \sim p_H(\mathbf{h}) \qquad \mathbf{x} = \mathbf{f}^{-1}(\mathbf{h}) \tag{3.6}$$

and that the determinant of the Jacobian $\det \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}$ is easy to compute so that we can evaluate the density p_X in (3.5).

For factorial prior p_H the data log likelihood is

$$\log p_X(\mathbf{x}) = \sum_{i=1}^{D} \log p_{H_i}(\mathbf{f}_i(\mathbf{x})) + \log \left| \det \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \right| , \tag{3.7}$$

which they call the *non-linear independent component estimation* criterion (NICE). The factorization in the prior should help the model to find meaningful structure in the data.

In analogy with VAE's they call $\mathbf{f}$ the *encoder* and $\mathbf{f}^{-1}$ the *decoder*.

3.2 Architecture

We can stack the layers of transformations $\mathbf{f}_L \circ \ldots \circ \mathbf{f}_2 \circ \mathbf{f}_1$ because we can get the inverse and determinants by layer and then stack them again (composition of inverses and product of per-layer Jacobian determinants).

3.2.1 Coupling layers

Main idea is to split the data vectors to blocks $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ use transformation $\mathbf{f}$

$$\mathbf{h}_1 = \mathbf{x}_1 \qquad \mathbf{h}_2 = \mathbf{g}(\mathbf{x}_2, \mathbf{m}(\mathbf{x}_1)) , \tag{3.8}$$

where $\mathbf{m}$ is the *coupling* function, an arbitrarily complex function such as NN.

The inverse $\mathbf{f}^{-1}$ is

$$\mathbf{x}_1 = \mathbf{h}_1 \qquad \mathbf{x}_2 = \mathbf{g}^{-1}(\mathbf{h}_2, \mathbf{m}(\mathbf{h}_1)) \tag{3.9}$$

and the Jacobian

$$\frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \frac{\partial \mathbf{h}_2}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{h}_2}{\partial \mathbf{x}_2} \end{bmatrix} \tag{3.10}$$

so that $\det \frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \det \frac{\partial \mathbf{h}_2}{\partial \mathbf{x}_2}$.

Additive coupling layer makes the inverse and determinant even easier. The additive transformation $\mathbf{f}$

$$\mathbf{h}_1 = \mathbf{x}_1 \qquad \mathbf{h}_2 = \mathbf{x}_2 + \mathbf{m}(\mathbf{x}_1) \tag{3.11}$$

has the inverse $\mathbf{f}^{-1}$

$$\mathbf{x}_1 = \mathbf{h}_1 \qquad \mathbf{x}_2 = \mathbf{h}_2 - \mathbf{m}(\mathbf{h}_1) \tag{3.12}$$

and Jacobian determinant $\det \frac{\partial \mathbf{h}}{\partial \mathbf{x}} = 1$.

They stack several coupling layers (4 but at least 3 needed) so that all dimensions may influence all the others.

In the last layer they further allow for rescaling by a diagonal scaling matrix so that each dimension is rescaled as $h_i^{(s)} = S_{ii} h_i$ with a simple Jacobian determinant $\prod_i S_{ii}$ If $S_{ii} \to \infty$ then the dimension of the latent space becomes irrelevant ($h_i^{(s)}/S_{ii} \to 0$). This is similar to non-linear PCA with the scaling S_{ii} as the eigenspectrum.

With the factorial prior, additive coupling layers and the rescaling in the last layer the NICE criterion becomes

$$\log p_X(\mathbf{x}) = \sum_{i=1}^{D} \Big(\log p_{H_i}(\mathbf{f}_i(\mathbf{x})) + \log |S_{ii}| \Big) . \tag{3.13}$$

They use simple isotropic Gaussian or logistic as the prior distribution.

3.3 Links to other methods

VAE has a stochastic encoder $q(\mathbf{h}|\mathbf{x})$ (while NICE has deterministic $\mathbf{h} = \mathbf{f}(\mathbf{x})$) and a noisy decoder $p(\mathbf{x}|\mathbf{h})$ which can be made deterministic by simply using the mean or mode of the distribution (similar to the NICE deterministic $\mathbf{x} = \mathbf{f}^{-1}(\mathbf{h})$). However, the sampling of $\mathbf{h}$ passed to the decoder injects noise into the auto-encoding architecture.

You can look at the two terms in the objective (3.7) as the maximization of the likelihood p_H of the code $\mathbf{h} = \mathbf{f}(\mathbf{x})$ and a regularization term $\log \left| \det \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} \right|$ encouraging the encoded distribution to occupy more volume, kind of similarly as in VAEs you encourage high entropy of the approximate posterior.

Obviously this is all linked to inverse transform sampling with the cumulative distribution function as the transformation.

GANs transform a noise variable to data $\mathbf{x} = \mathbf{g}(\mathbf{h})$ but do not ensure that the learned transformation $\mathbf{g}$ is invertibel so cannot evaluate and maximaze the data likelihood. Instead they use an adversary to discriminate between true and GAN samples of data.

4 Rezende et al.: Normalizing flows for spheres and tori

Paper: Danilo Jimenez Rezende, George Papamakarios, et al. "Normalizing Flows on Tori and Spheres". In: *arXiv:2002.02428 [cs, stat]* (2020)

Notes taken: 12/2/2020

TLDR: Normalizing flows over non-Euclidean data should take into account the particular topology of the space and use specific diffeomorphic transformations. If you project to $\mathbb{R}^D$ to do the flows there, the projection itself may not be diffeomorphic and therefore the whole chain of transformations in the change of variable formula breaks. Quite a lot of math bruhaha for modelling data on spheres and tori.

4.1 Intro

Use normalizing flows, that is the usual change of variable rule for density, e.g. equation (1.34) over data in some non-Euclidean space $\mathcal{M}$. The simple solution of applying the flow in $\mathbb{R}^D$ and then project to $\mathcal{M}$ is problematic when $\mathcal{M}$ is not diffeomorphic to $\mathbb{R}^D$.

The simple solution of applying the flow in $\mathbb{R}^D$ and then project to $\mathcal{M}$ is problematic if $\mathcal{M}$ and $\mathbb{R}^D$ are not diffeomorphic.

They focus on data on a sphere $\mathbb{S}^D$ or tori $\mathbb{T}^D$. A circle $\mathbb{S}^1$ in $\mathbb{R}^2$ can be parametrised either by $\{(x_1, x_2) \in \mathbb{R}^2 : x_1^2 + x_2^2 = 1\}$ or by $\theta \in [0, 2\pi]$. They give conditions for a valid diffeomorphism $f : [0, 2\pi] \to [0, 2\pi]$ taking into account the periodic nature of the circle. They then discuss three types of diffeomorphisms on circle: M obius transformations, circular splines and non-compact projections.

They explain how to combine these either by composition or as convex combinations to increase the expressivity of the transformed distributions and the complexity of evaluating the final transform f and its inverse f^{-1}. Sometimes these cannot be done analytically and require numerical methods.

Generalizations to torus are based on autoregressive flows over the circle transformers. To extend to higher-dimensional spheres they use recursive construction based on exponential map and cylindrical coordinates.

It is all very mathematical (space topology) and I'm not sure how useful. It seems they want to extend this to Li groups that or of interest in fundamental physics (particle interactions).

5 Rezende's variational inference with normalizing flows

Paper: Danilo Jimenez Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". In: *arXiv:1505.05770 [cs, stat]* (2016)

Notes taken: 13/2/2020

TLDR: Increase the flexibility of the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ in VAEs by constructing it as a normalizing flow $\mathbf{z} = f(\mathbf{z}_0)$ where $\mathbf{z}_0 \sim \mathcal{N}(0, I)$ and f is a neural network stacking a set of specifically designed transformations, planar or radial flows, which have relatively cheap Jacobian determinants. Here they never need the inverse f^{-1} so they don't care about the cost of inverting.

One shortcoming of standard VAEs is that if the class of approximate posterior distributions $q_\phi(\mathbf{z}|\mathbf{x})$ is too simple, such as isotropic Gaussian, this will be too far from the true posterior $p(\mathbf{z}|\mathbf{x})$ thus hampering the effectiveness of the inference.

They propose to use normalizing flows to increase the flexibility of the approximate posteriors. This is done through a standard stacking of differentiable invertible transformations starting from a simple latent distribution such as isotropic Gaussian.

They make a link of flows to the standard expectation identity $\mathbb{E}_{h(\mathbf{X})}(h(\mathbf{x})) = \mathbb{E}_{\mathbf{X}}(h(\mathbf{x}))$ and interpret the flows as a series of contractions and expansions of the initial density.

They call *infinitesimal flows* a stacking of transformations with the number of stacks tending to infinity. This can be described by partial differential equations describing the continuous-time dynamics.

They give two examples: *Langevin flow* and *Hamiltonian flow*. The first has been previously used for sampling from complex distributions, the second can be used to describe the dynamics of Hamiltonian MC. *This is beyond me.*

Constructing normalizing flows is in principle easy, but the computational complexity will be prohibitive for any reasonable application. The problem is mainly the computation of the Jacobian determinant and its gradients. They propose two classes of invertible transformations:

planar flows with contractions and expansions in direction perpendicular to a hyperplane $\mathbf{w}^T\mathbf{z} + b = 0$

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b) \ , \tag{5.1}$$

where h is an elementwise non-linearity and

radial flows with contractions and expansions around a reference point $\mathbf{z}_0$

$$f(\mathbf{z}) = \mathbf{z} + \beta\frac{\mathbf{z} - \mathbf{z}_0}{\alpha + |\mathbf{z} - \mathbf{z}_0|} \tag{5.2}$$

and show how to obtain the Jacobian determinant in linear time. They show on simple simulated example what these transformations look like starting from simple Gaussian.

Note: These transformations are only invertible under some conditions that they give in the appendix and show how to ensure these in training (e.g. re-parametrization). But the fact that they are invertible does not make them computationally easy to invert! Nevertheless, in their case it is not a problem cause they do not actually need the inverse for anything. They never need to evaluate the density of $\mathbf{x}$ which would not be constructed from $\mathbf{z}$ through $\mathbf{x} = f(\mathbf{z})$ so that they would need to recover $\mathbf{z} = f^{-1}(\mathbf{x})$.

They show how the standard ELBO objective can be updated to take into account the flow (the expectation of the Jacobian determinant appears) and how the whole VAE algorithm needs to be updated - just need to pass the sample z_0 through the transformation network and change accordingly the objective.

In experiments they compare to NICE (Dinh, Krueger, et al. 2015) (section 3) and show that they can get closer to the true distributions through lower number of transformations. When plugged to VAEs for MNIST, they get better log-likelihood values (estimated by importance sampling).

6 Tran et al.: Discrete flows

Paper: Dustin Tran et al. "Discrete Flows: Invertible Generative Models of Discrete Data". In: *Advances in Neural Information Processing Systems 32 (NIPS 2019)*. 2019

Notes taken: 18/2/2020

TLDR: Extending normalizing flows to discrete (categorical) distributions. For invertible $\mathbf{y} = f(\mathbf{x})$ the change-of-variable simplifies to

$$p_Y(\mathbf{y}) = p_X(f^{-1}(\mathbf{y})) \tag{6.1}$$

because discrete data have no volume that should be catered for by the Jacobian determinant. All this can achieve is relabel the categories so that the transformed variables are easier to model (e.g. become independent). Entropy of the original and transformed variable is the same.
The flow they propose follows from the autoregressive or bipartite constructions (coupling layer)

$$y_d = (\mu_d + \sigma_d x_d) \bmod K \tag{6.2}$$

with (μ, σ) being either

- autoregressive functions of previous dimensions or
- for the coupling approach $(0, 1)$ for a subset of dimensions and a function of these for the rest of the output dimensions

Because of the discreteness, the forward pass uses one-hot encoding of arg max over the K-category output of the network. For the backward it uses the straight-through gradient estimator based on the softmax approximation.
Experiments look reasonably good with comparative results to baselines but in the bipartite case much faster generations.

6.1 Intro

Normalizing flows based on the change-of-variable formula have not previously been explored for discrete variables.

Here they extend the ideas from the continous case to the discrete. They specifically foucs on two architectures

- discrete autoregressive flows with μ, σ defined as autoregressive functions

$$\begin{aligned}
\mathbf{y} = f(\mathbf{x}): &\qquad y_d = \mu_d + \sigma_d x_d, \quad (\mu_d, \sigma_d) = h(y_1, \dots, y_{d-1}), \quad d = 1, \dots, D \\
\mathbf{x} = f^{-1}(\mathbf{y}): &\qquad x_d = \sigma_d^{-1}(y_d - \mu_d), \quad (\mu_d, \sigma_d) = h(y_1, \dots, y_{d-1}), \quad d = 1, \dots, D \\
&\det \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{y}} = \det\left(\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\mathbf{x}}\right)^{-1} = \left(\det \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\mathbf{x}}\right)^{-1} = \prod_{i=1}^{D} 1/\sigma_i
\end{aligned} \tag{6.3}$$

- discrete bipartite flows with *coupling layers* as in the realNVPs (section 2)

$$\begin{aligned}
\mathbf{y} = f(\mathbf{x}): &\qquad \mathbf{y}_{1:d} = \mathbf{x}_{1:d}, \quad \mathbf{y}_{d+1:D} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \mathbf{x}_{d+1:D}, \quad (\boldsymbol{\mu}, \boldsymbol{\sigma}) = h(\mathbf{x}_{1:d}) \\
\mathbf{x} = f^{-1}(\mathbf{y}): &\qquad \mathbf{x}_{1:d} = \mathbf{y}_{1:d}, \quad \mathbf{x}_{d+1:D} = (\mathbf{y}_{d+1:D} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma}, \quad (\boldsymbol{\mu}, \boldsymbol{\sigma}) = h(\mathbf{x}_{1:d}) \\
&\det \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{y}} = \prod_{i=d+1}^{D} 1/\sigma_i
\end{aligned} \tag{6.4}$$

6.2 Discrete flows

In general for discrete r.v. $\mathbf{X} \sim p(\mathbf{x})$ and function $\mathbf{y} = f(\mathbf{x})$ the induced probability mass function of $\mathbf{Y}$ is

$$p(\mathbf{Y} = \mathbf{y}) = \sum_{\mathbf{x} \in f^{-1}(\mathbf{y})} p(\mathbf{X} = \mathbf{x}) \ , \tag{6.5}$$

where $f^{-1}(\mathbf{y})$ is the pre-image of $\mathbf{y}$ under f.

For invertible f this simplifies to

$$p(\mathbf{Y} = \mathbf{y}) = p(\mathbf{X} = f^{-1}(\mathbf{y})) \ . \tag{6.6}$$

Since for discrete distributions *volume* is not defined (counting measure vs Lebesgue measure), there is no need to account for it as in the continuous case via the Jacobian determinant (see e.g. equation (3.5)).

What discrete flows do is relabelling of the data so that the relabelled data distribution can be modelled with the base distribution. However, the number of categories remains the same and the entropy of the base and target distribution is preserved.

xor transform For example bi-variate distribution with dependent r.v. $\mathbf{x} = (x_1, x_2)$ can be relabelled into independet r.v. $\mathbf{y} = (y_1, y_2)$ using the xor $\oplus$ operator $(y_1, y_2) = (x_1, x_1 \oplus x_2)$

	$x_2 = 0$	$x_2 = 1$	
$x_1 = 0$|	0.63	0.07	|0.7
$x_1 = 1$|	0.03	0.27	|0.3
	0.66	0.34	

$$p(x_1, x_2) \neq p(x_1)p(x_2)$$

With the xor transform $\mathbf{y} = f(\mathbf{x}) = (y_1, y_2) = (x_1, x_1 \oplus x_2)$ and $\mathbf{x} = f^{-1}(\mathbf{y}) = (x_1, x_2) = (y_1, y_1 \oplus y_2)$ the transformed variable .

	$y_2 = 0$	$y_2 = 1$	
$y_1 = 0$|	0.63	0.07	|0.7
$y_1 = 1$|	0.27	0.03	|0.3
	0.9	0.1	

$$p(y_1, y_2) = p(y_1)p(y_2)$$

modulo location-scale transform for a D dimensional vector $\mathbf{x} = (x_1, \ldots, x_D)$ with each x_i taking values in $0, 1, \ldots, K-1$

$$y_d = (\mu_d + \sigma_d x_d) \bmod K, \quad (\mu_d, \sigma_d) = h(y_1, \ldots, y_{d-1}) \tag{6.7}$$

with autoregressive functions (μ_d, σ_d). (The bipartite version would have $(\mu, \sigma) = (0, 1)$ for a subset of the dimensions and being a function of these dimensions for outputting the other dimensions.)

For the transform to be invertible σ and K need to be *coprimes* which is possible to ensure by masking noninvertible values of σ or fixing $\sigma = 1$ and the inverse can be found by Euclid's algorithm *no clue what it is*.

The modulo transform does not preserve ordering information so not good for ordinal data.

6.2.1 Training

The flow is defined (parametrised) by the μ and σ autoregressive or bipartite networks and the parameters of the base distribution.

μ and σ should be discrete. It is achieved by the network spitting out two vectors of K logits θ_d for each dimension d and using for the forward pass

$$\boldsymbol{\mu}_d = \text{one\_hot}(\arg\max(\theta_d)) \ . \tag{6.8}$$

Since this is non-differentiable for the backward pass they use

$$\frac{\mathrm{d}\boldsymbol{\mu}_d}{\mathrm{d}\theta_d} \approx \frac{\mathrm{d}}{\mathrm{d}\theta_d}\text{softmax}\left(\frac{\theta_d}{\tau}\right) \tag{6.9}$$

with temperature fixed to $\tau = 0.1$ in experiments (which I understand is a version of the straight-through gradient estimation).

6.3 Experiments

In the experiments they use autoregressive categorical with reverse ordering as the base distribution for autoregressive flows and factorized categorical base for bipartite flows.

They show on some toy examples that 1 flow over base autoregressive improves over just the base autoregressive distribution an that bipartite flow is only slightly worse while much faster (parallel) for generations.

They also explored tasks such as addition of multiple digit integers (this in my head is more about exploring the strength of autoregressive flows for feature extraction than for generations), Potts model, and character-level language modelling (I don't know the datasets and cannot quite appreciate the beauty of it). In all of these, the bipartite performed reasonably well and was much faster to generate new examples.

7 Hoogeboom et al.: Discrete flows for compression

Paper: Emiel Hoogeboom et al. "Integer Discrete Flows and Lossless Compression". In: *Advances in Neural Information Processing Systems 32 (NIPS 2019)*. 2019

Notes taken: 18/2/2020

TLDR: The idea is to use invertible flows to learn discrete distribution over digitized data (data in discrete integer space) to be able to efficiently compress using an entropy encoder over the latent $\mathbf{z} = f_{ID}(\mathbf{x})$, where f_{ID} is the Integer Discrete Flow.

The encoder doing the compression is of-the-shelf encoder (not the aim of the paper) based on the entropy principals (optimal encoding length is the Shannon information content $c = enc(z), l(c) = -\log_2 p(z)$. The important thing is to learn correctly the $p(z)$ (maximize the likelihood) so that the encoder uses the best length distribution.

The IDF they propose build on RealNVP combining coupling and factor-out layers with the coupling being simple additive $\mathbf{z} = [\mathbf{z}_a, \mathbf{z}_b] = f_{ID}(\mathbf{x}) = [\mathbf{x}_a, \mathbf{x}_b + round(\mathbf{t}(\mathbf{x}_a))]$ with rounding to ensure that they stay in integers (and no scaling to ensure the co-domain is the same as the domain).
Otherwise they use all the squeeze and permutation tricks of RealNVP.

7.1 Intro

Based on (MacKay 2005):

Lossless compression preserves information perfectly. A naive way of encoding a discrete r.v. X with possible outcomes (categories) $\{a_1, \dots, a_K\}$. If we assign a unique binary code to each outcome, then the length of the codes would be the **raw bit content** of X.

$$H_0(X) = \log_2 K \tag{7.1}$$

Naturally, we wish to encode the data to shortest messages possible. There is no way we can make all codes shorter but we can make some codes shorter and some longer so that on average (in expectation), the transmitted messages will be shorter. This can be achieved if the most frequent codes (those occurring with higher probability) will be shorter and less frequent codes (those occurring with lower probability) will be longer.

Definition 7.1 Assume a random variable $X \sim p(a_i)$ generated from a discrete probability distribution $p(a_i), i = 1, \dots, K$ with K possible outcomes (categories). The **expected message length** $L(C, X)$ of encoding C of the r.v. $X \sim p(a_i)$ is $L(C, X) = \sum_{i=1}^{K} p(a_i) l(a_i)$, where $l(a_i)$ is the length of the encoding of the symbol (category) a_i.

The expected message length is lower-bounded by the entropy of X so that $L(C, X) \geq \mathrm{H}(X) = -\sum_{i=1}^{K} p(a_i) \log_2 p(a_i)$ and it is optimised (minimised) if the encoding lengths are equal to the Shannon information content of the outcomes $l(a_i) = -\log_2 p(a_i)$.

If we use an encoder which assigns different lengths to the codes

$$l_q(a_i) = -\log_2 q(a_i) \quad \Longleftrightarrow \quad 2^{-l_q(a_i)} = q(a_i) \qquad i = 1, \dots, K \ , \tag{7.2}$$

where $q(a_i)$ is the implicit probability distribution of the encoding. The expected message length of such an encoding is always greater than of the optimal encoding using the Shannon information content to define the lengths

$$L(C_q, X) = -\sum_{i=1}^{K} p(a_i) \log_2 q(a_i) = -\sum_{i=1}^{K} p(a_i) \log_2 \frac{q(a_i)p(a_i)}{p(a_i)}$$
$$= -\sum_{i=1}^{K} p(a_i) \log_2 p(a_i) + \sum_{i=1}^{K} p(a_i) \log_2 \frac{p(a_i)}{q(a_i)} = \mathrm{H}(X) + \mathrm{KL}(p \parallel q) \tag{7.3}$$

exceeding the optimal expected encoding length (the entropy $\mathrm{H}(X)$) by the relative entropy (KL divergence) $\mathrm{KL}(p \parallel q)$.

Back to Hoogeboom:

To be able to encode random data X efficiently, we need to learn the distribution $p(a_i)$. Maximizing the (log) likelihood $p(X)$ of the data X is thus equivalent to minimizing the expected message length.

The naive idea of estimating the data distribution by simply counting the frequencies (as in a histogram) breaks down in high dimensions. Deep generative models allow for learning complicated data distributions. Flow based models have advantage over other deep generative models

- they admit exact log-likelihood estimation (unlike VAEs)
- sampling (decoding) is relatively cheap ãs expensive as inference (unlike in PixelCNNs)

Standard flow models f are defined for continuous variables. If these were to be used for compression, the latent code $z = f(x)$ would have to be quantized (discretized). Instead they propose to work directly over the digital *ordinal discrete data* such as images with 256 values per pixel, and digital video or audio to remain in the discrete domain by using **integer discrete flows** (IDF).

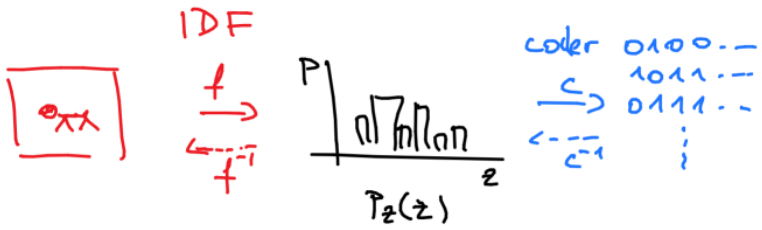


Figure 1: IDB-based lossless compression: first the data $x \sim p_x$ is passed through the learned IDF $z = f_{id}(x)$ to get to the latent $z \sim p_z$ with known distribution p_z, then z is encoded by any off-the-shelf entropy-based encoder (e.g. Huffman, or arithmetic) into the bitstream c. For getting x from the bistream c, it is first decoded (off-the-shelf) to obtain z and then the inverse IDF maps it to $x = f_{id}^{-1}(z)$.

7.2 Standard flow layers

Normalizing flows is a stack of density transformations by the change-of-variable formula (e.g. equation (3.5)) using bijective transformations $f_i : \mathcal{X}_i \to \mathcal{X}_{i+1}$.

They consider:

- **coupling layers** as in realNVPs (section 2) with splitting $\mathbf{x} = [\mathbf{x}_a, \mathbf{x}_b]$

$$\begin{aligned} \mathbf{z} = [\mathbf{z}_a, \mathbf{z}_b] = f(\mathbf{x}) &= [\mathbf{x}_a, \mathbf{t}(\mathbf{x}_a) + \mathbf{s}(\mathbf{x}_a) \odot \mathbf{x}_b], \quad \mathbf{s}(\mathbf{x}_a) \neq 0 \\ \mathbf{x} = [\mathbf{x}_a, \mathbf{x}_b] = f^{-1}(\mathbf{z}) &= [\mathbf{z}_a, (\mathbf{z}_b - \mathbf{t}(\mathbf{z}_a)) \oslash \mathbf{s}(\mathbf{x}_a)] \\ \det \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{z}} = \prod_i \mathbf{s}(\mathbf{x}_a)_i^{-1} \qquad & p_x(\mathbf{x}) = p_z(f(\mathbf{x})) \prod_i \mathbf{s}(\mathbf{x}_a)_i^{-1} \end{aligned} \tag{7.4}$$

- **factor-out layers** also as in realNVPs (section 2). Here the idea is that not all the dimensions need to be propagated through all the flow layers. Instead part of the dimensions may be *factored-out* at regular intervals and the rest of the flow operates over the remaining lower dimensional data. For example for two layers we get

$$\begin{gathered} \mathbf{z} = [\mathbf{z}_1, \mathbf{z}_2]: \quad [\mathbf{z}_1, \mathbf{y}_1] = f_1(\mathbf{x}) \qquad \mathbf{z}_2 = f_2(\mathbf{y}_1) \\ \mathbf{x} = f_1^{-1}\left(\mathbf{z}_1, f_2^{-1}(\mathbf{z}_2)\right) \\ p(\mathbf{z}_1, \mathbf{y}_1) = p(\mathbf{y}_1) p(\mathbf{z}_1|\mathbf{y}_1) \qquad p_y(\mathbf{y}_1) = p_{z_2}(f_2(\mathbf{y}_1)) \left|\frac{\mathrm{d}f_2(\mathbf{y}_1)}{\mathrm{d}\mathbf{y}_1}\right| \\ p(\mathbf{x}) = p(\mathbf{z}_1, \mathbf{y}_1)\left|\frac{\mathrm{d}f_1(\mathbf{x})}{\mathrm{d}\mathbf{x}}\right| = p(\mathbf{y}_1) p(\mathbf{z}_1|\mathbf{y}_1)\left|\frac{\mathrm{d}f_1(\mathbf{x})}{\mathrm{d}\mathbf{x}}\right| = p_{z_2}(f_2(\mathbf{y}_1)) \left|\frac{\mathrm{d}f_2(\mathbf{y}_1)}{\mathrm{d}\mathbf{y}_1}\right| p(\mathbf{z}_1|\mathbf{y}_1)\left|\frac{\mathrm{d}f_1(\mathbf{x})}{\mathrm{d}\mathbf{x}}\right| \end{gathered}$$

  This allows for conditional dependence between parts of $\mathbf{z}$

$$p(\mathbf{z}) = p(\mathbf{z}_L) p(\mathbf{z}_{L-1}|\mathbf{z}_L) p(\mathbf{z}_{L-2}|\mathbf{z}_L, \mathbf{z}_{L-1}) \ldots p(\mathbf{z}_1|\mathbf{z}_2, \ldots, \mathbf{z}_L) \tag{7.5}$$

7.2.1 Integer discrete flows (IDF)

For integer-valued observations $\mathbf{x} \in \mathcal{X} \in \mathbb{Z}^d$ and prior distribution with $p_Z(.)$ with support on $\mathbb{Z}^d$ consider a bijection $f : \mathbb{Z}^d \to \mathbb{Z}^d$ so that

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}), \qquad \mathbf{z} = f(\mathbf{x}) \tag{7.6}$$

IDF stacks multiple such layers.

Integer discrete coupling They design the layers to ensure that the IDF map is closed on $\mathbb{Z}^d$ so that they don't have to keep track of the co-domain of f_{ID}. For $\mathbf{x} = [\mathbf{x}_a, \mathbf{x}_b] \in \mathbb{Z}^d$

$$\begin{aligned} \mathbf{z} = [\mathbf{z}_a, \mathbf{z}_b] &= f_{ID}(\mathbf{x}) = [\mathbf{x}_a, \mathbf{x}_b + round(\mathbf{t}(\mathbf{x}_a))] \\ \mathbf{x} = [\mathbf{x}_a, \mathbf{x}_b] &= f_{ID}^{-1}(\mathbf{z}) = [\mathbf{z}_a, \mathbf{z}_b - round(\mathbf{t}(\mathbf{x}_a))] \end{aligned} \tag{7.7}$$

They split $a : b$ as 75% : 25% to apply the rounding to fewer dimensions and have less gradient bias. The trainable parameters are within the $\mathbf{t}$ function (a network). To back-propagate through the rounding they use straight through estimator with $\nabla_x(x) = I$.

As the prior distribution $p_Z(.)$ they use the discretized logistic $DLogistic(z|\mu, \sigma)$ which assigns to an integer $z \in \mathbb{Z}$ the probability over the interval $[z - 0.5, z + 0.5]$ under the logistic distribution.

$$DLogistic(z|\mu,\sigma) = \int_{z-0.5}^{z+0.5} Logistic(z'|\mu,\sigma) = F_{logistic}(z{+}0.5) {-} F_{logistic}(z{-}0.5) = \sigma(\frac{z+0.5-\mu}{\sigma}) {-} \sigma(\frac{z-0.5-\mu}{\sigma})$$

In the factor-out set-up we have for each factoring layer l $p(z_{li}|y_{li}) = DLogistic(z_{li}|\boldsymbol{\mu}(y_l)_i, \boldsymbol{\sigma}(y_l)_i)$ where $\boldsymbol{\mu}, \boldsymbol{\sigma}$ are outputs of neural networks (in the implementation it is just one network spitting out μ and log scale vectors).

The latent in the last layer z_L has an unconditional learned prior $DLogistic(z_{Li}|\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i)$, where $\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i$ are trainable parameters appearing in the loss through the likelihood evaluation. They also consider a mixture of these where they learn an additional mixing parameter π_i (they work with 5 component mixtures.)

7.3 Architecture

They split the archi into levels, where each level has a *squeezing operation*, $Dintegerflows$ and factor-out layer. Each integer flow consists of permutation (to ensure that all dimensions can interact with all other) followed by a discrete coupling layer. The permutations are initialised once and then kept fixed (not trained).

There is a trade-off between complexity and gradient bias influenced by the number of rounding operations. Instead of increasing the number of IDFs they make more complex the $\mathbf{t}, \boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ parts of the network.

7.4 Experiments

They train IDF over ImageNet32, ImageNet64 and CIFAR10 and then evaluate the likelihood and compression rate over test data ($x \to z = f_{id}(x) \to c = encode(z) \to z' = decode(c) \to x' = f_{id}^{-1}(z')$). I'm guessing the bits per dimension they mention is the log_2 likelihood values per pixel d in the input space $\mathbb{Z}^d$ evaluated over x' reconstructed from the test data x. The compression rate is probably the length of encodings c of the test data compared to some naive encoding c'. Perhaps $L(c', x) = \log_2 n_{test}$?

It seems to beat the standard baselines such as JPEG, PNG, FLIF and Bit-Swap in terms of both, the log-likelihood as well as compression rate. It even performs well if IDF is trained on other data then the the target distribution (e.g. IDF trained on ImageNet32 and used for ImageNet64 and CIFAR10) datasets.

They also have good results for compressing large images by smaller patches and in progressive decompression.

See next page for my understanding of the implementation in their github repository.

7.5 My pseudo-code for implementation

What I understood from the pyTorch implementation:
For the **forward pass** you:

1. take $\mathbf{x}$
2. $\mathbf{z} \leftarrow squeeze(\mathbf{x})$ (fixed transform)
3. repeat D times
   - $\mathbf{z} \leftarrow permute(\mathbf{z})$ (fixed transform)
   - $\mathbf{z} \leftarrow coupling(\mathbf{z})$ ($[\mathbf{z}_a, \mathbf{z}_b + \mathbf{t}(\mathbf{z}_a)]$ where $\mathbf{t}()$ are trained neural networks, one for each D, eating on $\mathbf{z}_a$ $\mathbf{z}$ and spitting out an output with the same dimensions as $\mathbf{z}_b$)
4. $[\mathbf{z}, \mathbf{y}] \leftarrow split(\mathbf{z})$ (split $\mathbf{z}$ for factoring-out parts of the dims, you won't transform $\mathbf{y}$ any more, $\mathbf{z}$ will be further transformed)
5. $\mu, \sigma \leftarrow h(\mathbf{z})$ where h is a nn with outputs μ, σ. These are the parameters of the conditional distribution $p(\mathbf{y}|\mathbf{z})$ (careful, here the notation is swapped as compared to the description in the paper).
6. evaluate $lpy = \log p(\mathbf{y}|\mu(\mathbf{z}), \sigma(\mathbf{z}))$ (this will enter the loss)
7. $\mathbf{x} \leftarrow \mathbf{z}$ and goto 2 and repeat N-1 times
8. on the Nth repetition skip 4-6
9. evaluate $lpz = \log p(\mathbf{z}|\mu, \sigma)$ where μ, σ are learned parameters. These are learned by maximizing the objective function OF
10. evaluate the loss: $OF = lpz + \sum lpy$ (the OF is equal to $\log p(\mathbf{x})$ when you realize that $\mathbf{z}$ and $\mathbf{y}$'s are different dimensions of a single latent variable living in $\mathbb{Z}^d$, the same as $\mathbf{x} \in \mathbb{Z}^d$.)

Train by standard backpropagation. Once trained, you can pass in an input $\mathbf{x}$ and the OF is its log-likelihood - *you can evaluate the likelihood of your data points*.

For **sampling** you:

1. sample $\mathbf{z}$ $p(\mathbf{z}|\mu, \sigma)$, where μ, σ are the trained (now fixed parameters)
2. repeat D times
   - $\mathbf{z} \leftarrow inversecoupling(\mathbf{z})$ ($[\mathbf{z}_a, \mathbf{z}_b - \mathbf{t}(\mathbf{z}_a)]$ where $\mathbf{t}()$ are the trained neural networks)
   - $\mathbf{z} \leftarrow inversepermute(\mathbf{z})$ (fixed transform)
3. $\mathbf{z} \leftarrow unsqueeze(\mathbf{x})$ (fixed transform)
4. get parameters $\mu(\mathbf{z}), \sigma(\mathbf{z}) \leftarrow h(\mathbf{z})$ where h is the trained nn
5. sample $\mathbf{y} \sim p(\mathbf{y}|\mu(\mathbf{z}), \sigma(\mathbf{z}))$
6. stack $[\mathbf{z}, \mathbf{y}]$ and go to 2 and repeat N times (perhaps N-1 times?)

Once you have done this N times the length of your stackings $[\mathbf{z}, \mathbf{y}_1, \ldots, \mathbf{y}_N]$ should be the same as of $\mathbf{x}$ and you will get a sample of the data $\mathbf{x}$ by performing the last inverse coupling, inverse permutaions and unsqueezing.

Following the above pseudo-algo, I should be able to implement this. ;)

8 Kingma's VAE with inverse autoregressive flows

Paper: Durk P Kingma et al. "Improved Variational Inference with Inverse Autoregressive Flow". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. 2016

Notes taken: 22/2/2020

TLDR: The aim is to increase the flexibility of the approximate posterior (the inference model) $q(\mathbf{x}|\mathbf{x})$ in standard VAEs through normalizing flows on $\mathbf{z}$.

Standard Guassian autoregressive models build the random variable $\mathbf{y} = \{y_i\}_{i=1}^D$ by fixing an order in the dimensions and then getting the dimensions one by one as transformation of some easy to sample random variable $\boldsymbol{\epsilon} = \{\epsilon_i\}_{i=1}^D$ (this is not the data here!).

$$\mathbf{y} = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon} \tag{8.1}$$

$$y_0 = \mu_0 + \sigma_0\,\epsilon_0, \quad y_i = \mu_i(\mathbf{y}_{1:i-1}) + \sigma_i(\mathbf{y}_{1:i-1})\,\epsilon_i,\ i = 1,\dots D-1 \tag{8.2}$$

with not-autoregressive μ_0, σ_0 and μ_i, σ_i being the outputs of some autoregressive network. The problem with this is that the sampling is slow (you need to sample recursively). Hence this is not very practical in VAEs with high dimensional latent space $\mathbf{z}$ because there we need to sample the $\mathbf{z}$ every time we do a forward pass for each of the inputs $\mathbf{x}$.

The *inverse autoregressive flows* (IAF) say you should flip the dependency so that

$$\mathbf{y} = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon} \tag{8.3}$$

$$y_0 = \mu_0 + \sigma_0\,\epsilon_0, \quad y_i = \mu_i(\boldsymbol{\epsilon}_{1:i-1}) + \sigma_i(\boldsymbol{\epsilon}_{1:i-1})\,\epsilon_i,\ i = 1,\dots D-1 \tag{8.4}$$

This is much faster to sample (can be parallelized) and the Jacobian is still triangular with simple terms on the diagonal (just $\sigma_i(\boldsymbol{\epsilon}_{1:i-1})$) so that the induced density $p(\mathbf{y})$ is easy to evaluate.

8.1 Intro

The aim is to improve VAE's by increasing the flexibility of the inference model $q(\mathbf{z}|\boldsymbol{\epsilon})$ (so that the approximate posterior $q(\mathbf{z}|\boldsymbol{\epsilon})$ can be closer to the true posterior $p(\mathbf{z}|\boldsymbol{\epsilon})$) through *inverse autoregressive flows* (IAF).

8.1.1 Normalizing flows

Build posterior $q(\mathbf{z}|\boldsymbol{\epsilon})$ by starting off an initial random variable $\mathbf{z}_0$ with known density and distribution which is easy to sample from and apply a chain of invertible transformations f_t such that the last iterate $\mathbf{z}_T$ has more flexible distribution

$$\mathbf{z}_0 \sim q(\mathbf{z}_0|\boldsymbol{\epsilon}), \qquad \mathbf{z}_t = f_f(\mathbf{z}_{t-1,\boldsymbol{\epsilon}}),\ t = 1,\dots,T \tag{8.5}$$

$$\log q(\mathbf{z}_T|\boldsymbol{\epsilon}) = \log q(\mathbf{z}_0|\boldsymbol{\epsilon}) - \sum_{t=1}^{T} \log \left| \det \frac{\mathrm{d}\mathbf{z}_t}{\mathbf{z}_{t-1}} \right| \tag{8.6}$$

8.2 Inverse autoregressive transformation

The idea is based on Gaussian autoregressive functions. The variable $\mathbf{y} = \{y_i\}_{i=1}^D$ is modelled as a transformation of an easy to sample random variable $\boldsymbol{\epsilon} = \{\epsilon_i\}_{i=1}^D$ as

$$\mathbf{y} = \boldsymbol{\mu}(\mathbf{y}) + \boldsymbol{\sigma}(\mathbf{y})\boldsymbol{\epsilon} \ , \tag{8.7}$$

where $\boldsymbol{\mu}(\mathbf{y})$ and $\boldsymbol{\sigma}(\mathbf{y})$ denote the autoregressive functions so that

$$y_i = \mu_i(\mathbf{y}_{1:i-1}) + \sigma_i(\mathbf{y}_{1:i-1})\,\epsilon_i, \quad y_0 = \mu_0 + \sigma_0\,\epsilon_0 \ . \tag{8.8}$$

The computational cost of sampling is proportional to the dimension D as we need to iteratively follow from the ancestral dimensions. However, the determinant of the Jacobian $\frac{\partial \mathbf{y}}{\partial \mathbf{y}}$ is trivial. The Jacobian is lower triangular with zeros on the diagonal because

$$\frac{\partial[\mu_i(\mathbf{y}_{1:i-1}), \sigma_i(\mathbf{y}_{1:i-1})]}{\partial y_j} = [0, 0] \quad \text{for } i \leq j \ . \tag{8.9}$$

The inverse transformation

$$\boldsymbol{\epsilon} = \frac{\mathbf{y} - \boldsymbol{\mu}(\mathbf{y})}{\boldsymbol{\sigma}(\mathbf{y})} \qquad \epsilon_i = \frac{y_i - \mu_i(\mathbf{y}_{1:i-1})}{\sigma_i(\mathbf{y}_{1:i-1})} \tag{8.10}$$

can be panellized for sampling since the computation of dimensions of $\boldsymbol{\epsilon}$ does not depend on each other. This transformation has a simple Jacobian determinant as the Jacobian $\partial\boldsymbol{\epsilon}/\partial\mathbf{y}$ is lower triangular with $\partial\epsilon_i/\partial y_j = 0$ for $j > i$ and a simple diagonal $\partial\epsilon_i/\partial y_i = 1/\sigma_i(\mathbf{y}_{1:i-1})$. The log determinant is hence simply

$$\log\left|\det \frac{\partial\boldsymbol{\epsilon}}{\partial\mathbf{y}}\right| = -\sum_{i=1}^{D} \log\sigma_i(\mathbf{y}_{1:i-1}) \tag{8.11}$$

8.3 Inverse autoregressive flows

They bring the inverse transform into the VAE's as follows:

- let the encoder spit out initial $\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0, \mathbf{h}$
- initialize chain by $\mathbf{z}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\sigma}_0\boldsymbol{\epsilon}$, with $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \mathbf{I})$
- construct flow as the chain of transformations $\mathbf{z}_t = \boldsymbol{\mu}_t(\mathbf{z}_{t-1}, \mathbf{h}) + \boldsymbol{\sigma}_t(\mathbf{z}_{t-1}, \mathbf{h})\mathbf{z}_{t-1}$ with autoregressive $\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t$ outputs of an autoregressive neural network.

Jacobians of $\partial[\boldsymbol{\mu}_t(\mathbf{z}_{t-1}, \mathbf{h}), \boldsymbol{\sigma}_t(\mathbf{z}_{t-1}, \mathbf{h})]/\partial\mathbf{z}_{t-1}$ are triangular with zeros on the diagonal so that the Jacobian of each of the transformation is triangular with $\partial\mathbf{z}_t/\partial\mathbf{z}_{t-1} = \boldsymbol{\sigma}_t(\mathbf{z}_{t-1}, \mathbf{h})$ on the diagonal.

For gaussian $\boldsymbol{\epsilon}$ the density of the final iterate is

$$p(\mathbf{z}_T|\boldsymbol{\epsilon}) = -\sum_{i}^{D}\left(0.5\epsilon_i^2 + 0.5\log(2\pi) + \sum_{t=0}^{T}\log\sigma_{t,i}(\mathbf{z}_{t-1})\right) \tag{8.12}$$

Is this a Gaussian distribution with diagonal covariance? It should not be but the density looks like it is.

8.4 Implementation tricks

They use MADE (Germain et al. 2015) and PixelRNN (Aäron van den Oord et al. 2016) as the autoregressive networks for the flow. They also use a numerically more stable version of the flow transformation

$$[\mathbf{m}_t, \mathbf{s}_t] \leftarrow autoregressNN(\mathbf{z}_t, \mathbf{h}) \quad \boldsymbol{\sigma}_t = sigmoid(\mathbf{s}_t) \quad \mathbf{z}_t = \boldsymbol{\sigma}_t \odot \mathbf{z}_{t-1} + (1 - \boldsymbol{\sigma}_t) \odot \mathbf{m}_t \ . \tag{8.13}$$

I'm not sure of the role of $\mathbf{h}$ *which is being used as an input into all the flow layers.*

8.5 Experiments

They compare to baselines (results taken from other papers) over MNIST and CIFAR and show that likelihood (by importance sampling) is competitive and the generations can be much faster than that of standard autoregressive models such as PixelCNN.

9 Ardizzone: Inverse problems through invertible networks

Lynton Ardizzone, Jakob Kruse, et al. "Analyzing Inverse Problems with Invertible Neural Networks". In: *arXiv:1808.04730 [cs, stat]* (2019)

Notes taken: 24/2/2020

TLDR: You know from some physical model the forward function $\mathbf{y} = s(\mathbf{x})$ which allows you to simulate data. However, this does not mean that from the observations of $\mathbf{y}$ you can recover $\mathbf{x}$ (the inverse problem). They propose to learn the inverse function f^{-1} via invertible network ensuring that the learned forward approximates the simulator $f \approx s$.

The simulator may not be invertible (mathematically). Instead of learning a deterministic transformation $\mathbf{x} = f^{-1}(\mathbf{y})$ they therefore learn a probabilistic mapping $q(\mathbf{x}|\mathbf{y})$ where the conditional distribution is implied by a deterministic function $g(\mathbf{y}, \mathbf{z}; \theta)$ with a random variable $\mathbf{z} \sim \mathcal{N}(0, I)$.

They pose it into the invertible architecture so that they get $g^{-1}(\mathbf{x}) = f(\mathbf{x}; \theta) = [f_y(\mathbf{x}; \theta), f_z(\mathbf{z}; \theta)]$ with single parametrization. They use the standard change of variable formulas to recover the density $q(\mathbf{x}|\mathbf{y})$ and optimize the forward pass as well as the backward pass with a different losses (checking the prediction of $\mathbf{y}$ compared to the ground truth and checking the normality of $\mathbf{z}$ and its independence from $\mathbf{y}$ respectively.)

If you know the *forward function* (e.g. simulator) $\mathbf{y} = s(\mathbf{x})$ which generates some observables $\mathbf{y}$ from some hidden parameters, it does not mean we also know the inverse function s^{-1} to get the hidden parameters $\mathbf{x}$ from the observed $\mathbf{y}$.

Nice example: $\mathbf{x}$ are the parameters of a robotic arm (vertical location along a rail and 3 angles in 3 joints), $\mathbf{y}$ is the location (in 2D) where the arm reached. I can easily model the arm and get the location $\mathbf{y} = s(\mathbf{x})$. However, it is not obvious to get the setting of the parameters $\mathbf{x}$ for a known location $\mathbf{y}$. Or rather, there are many possible and what they want to infer is the posterior distribution $p(\mathbf{x}|\mathbf{y})$ over these.

I find this a bit ill posed because in the set-up above, there is no reason why there should be some other than uniform distribution over all possible configurations. Hm, but in the marginals perhaps some configurations can be more probable because they can combine with multiple configurations of the other parameters? And there is certainly plenty of dependencies.

They formulate it as a flow network where they

- assume latent variable model with $\mathbf{z} \sim p(\mathbf{z}) = \mathcal{N}(0, I_K)$ and $\mathbf{x} = g(\mathbf{y}, \mathbf{z}; \theta)$ where we learn the deterministic function g as a neural network
- by the standard change of variable we get for the conditional density

$$q(\mathbf{x}|\mathbf{y}) = p(\mathbf{z}) \left| \det J_x \right|^{-1}, \quad J_x = \left. \frac{\partial g(\mathbf{y}, \mathbf{z}; \theta)}{\partial [\mathbf{y}, \mathbf{z}]} \right|_{\mathbf{y}, f_z(\mathbf{x})} \tag{9.1}$$

- approximate the known forward function by learned $f_y(\mathbf{x}; \theta) \approx s(\mathbf{x})$ and learn it jointly with the inverse process

$$[\mathbf{y}, \mathbf{z}] = [f_y(\mathbf{x}; \theta), f_z(\mathbf{x}; \theta)] = f(\mathbf{x}; \theta) = g^{-1}(\mathbf{x}) \tag{9.2}$$

  tying together the parameters θ by the architecture of the invertible network.

There will be some problems with the dimensionality of the $\mathbf{x}$ and $[\mathbf{y}, \mathbf{z}]$ which normally should match but they propose to solve it simply by padding (and some more hacks, I think).

They then use a variant of the *coupling layers*, actually also interesting because it does not pass part of the inputs unchanged. Instead it changes everything but needs to do the updates and the inverses sequentially

$$\mathbf{v}_1 = \mathbf{u}_1 \odot exp(s_2(\mathbf{u}_2)) + t_2(\mathbf{u}_2), \qquad \mathbf{v}_2 = \mathbf{u}_2 \odot exp(s_1(\mathbf{v}_1)) + t_1(\mathbf{v}_1) \tag{9.3}$$

with the inverse

$$\mathbf{u}_2 = (\mathbf{v}_2 - t_1(\mathbf{v}_1)) \odot exp(-s_1(\mathbf{v}_1)), \qquad \mathbf{u}_1 = (\mathbf{v}_1 - t_2(\mathbf{u}_2)) \odot exp(-s_2(\mathbf{u}_2)) \tag{9.4}$$

The loss is not a simple maximum likelihood. Instead they do bi-derctional training where they accumulate the gradients from the forward and backward iterations and corresponding losses and only update after.

- For the forward iteration the loss penalizes deviations of the network predictions from the simulations $L_y(f_y(\mathbf{x}), s(\mathbf{x}))$. This can be something like squared error or cross entropy.
- For the backward iteration the loss penalizes the mismatch between the joint latent distribution and the product of their marginals $L_z(q(\mathbf{y}, \mathbf{z}), p(\mathbf{y}), p(\mathbf{z})) = MMD(q(\mathbf{y}, \mathbf{z}), p(\mathbf{y}), p(\mathbf{z}))$, where $p(\mathbf{z})$ is the normal prior, $p(\mathbf{y} = s(\mathbf{x})) = p(\mathbf{x}) \left| \det J_s \right|^{-1}$ with $J_s = \partial s(\mathbf{x}) / \partial \mathbf{x}$ and $q(\mathbf{y} = f_y(\mathbf{x}), \mathbf{z} = f_z(\mathbf{z})) = p(\mathbf{x}) \left| \det J_f \right|^{-1}$ with $J_f = \partial f(\mathbf{x}) / \partial \mathbf{x}$. Here $p(\mathbf{x})$ is some fixed prior distribution. This ensures that $\mathbf{y}, \mathbf{z}$ are independent and that the generated $\mathbf{z}$ follows the normal distribution. As the distance measure MMD they use the *Maximum Mean Discrepency* which can be evaluated efficiently over data samples
- To speed up convergence they add a third loss $L_x(p(\mathbf{x}), q(\mathbf{x})) = MMD(p(\mathbf{x}), q(\mathbf{x}))$, where $q(\mathbf{x}) = p(\mathbf{y} = f_y(\mathbf{x}))p(\mathbf{z} = f_z(\mathbf{x})) \left| \det J_x \right|^{-1}$ with J_x as in equation (9.1).

The experiments compare the MAP estimates of the $\mathbf{x}$ as well as the distributions through *calibration error* (something to do with the proportion of true data within a confidence interval) and they plot the marginal posterior densities to show how they deviate from the priors $p(\mathbf{x})$ (the kernel density estimate over the data?)

10 Kumar et al.: VideoFlow

Manoj Kumar et al. "VideoFlow: A Conditional Flow-Based Model for Stochastic Video Generation". In: (2020)

Notes taken: 24/6/2020

TLDR: They use flow model to predict future video frames. They argue by the usual advantages of flows as compared to VAEs, GANs and autoregressive (explicit likelihood, easier training, faster generations). The flows operate over individual frames (usual real-NVP and Glow architecture and tricks) and the temporal dependence is only factored into the latent distribution $p_\theta(\mathbf{z})$ via temporal autoregressive probability model.

The aim is stochastic prediction of video sequences - synthesizing RGB video frames conditioned on a few previous frames. Unlike previous work for video generations the flow-based model can generate diverse stochastic futures (as opposed to deterministic models) relatively cheaply (as compared to autoregressive models) and at the same time provide exact likelihood estimates (as opposed to VAEs or GANs).

Points out standard disadvantages of other generative models (VAEs & GANs lack of likelihood estimtes, GANs difficult to train, autoregressive slow to sample/generate). Moreover previously proposed *deterministic* models for video generation can only generate single future which either disregards other possibilities or mixes all of them into a blurry single future.

They use flow model for each frame with architectural tricks similar to Glow (section 13.5) with multi-scale architecture as in realNVP (section 2). To account for the time-dependency between frames they make the prior over the latent $\mathbf{z}$ autoregressive conditioning on the previous frames

$$p_\theta(\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{z}_t|\mathbf{z}_{<}t) \tag{10.1}$$

$$p_\theta(\mathbf{z}_t|\mathbf{z}_{<t}) = \prod_{l=1}^{L} p_\theta(\mathbf{z}_t^{(l)}|\mathbf{z}_{<t}^{(l)}, \mathbf{z}_t^{(>l)}) = \mathcal{N}(\mathbf{z}_t^{(l)}; \mu, \sigma) \tag{10.2}$$

$$(\mu, \log\sigma) = \text{neural net}_\theta(\mathbf{z}_{<t}^{(l)}, \mathbf{z}_t^{(>l)}) \tag{10.3}$$

where $< t$ indicates previous time frame, $> l$ indicates higher levels at the multi-scale architecture and the neural net is a 3D residual network *see Fig10 in paper for sketch of architecture*. This means that the temporal dependency between the video frames is factored into the model by the autoregressive structure in the latent prior while the flow model acts on individual frames. They claim making the flow temporal via 3D convolutions was too expensive to train.

The experiments show comparable or slightly better results compared to stochastic VAE or GAN based models in video prediction in terms of Frechet Video Distance, better in likelihood. However, they are quite honest in the evaluation showing also less good results such as accuracy of best prediction and failures for more complex dataset such as human motion videos (in the appendix).

They typically condition on only 1-3 previous frames and then generate up to 100 future frames. I wonder why they condition on so little data, I'm guessing computational complexity in combination with these datasets being such that only few frames give enough info for the future. Perhaps true for most datasets, needs some thinking.

11 Winkler et al.: Conditional Normalizing Flow

Christina Winkler et al. "Learning Likelihoods with Conditional Normalizing Flows". Version 1. In: (2019)

Notes taken: 2/7/2020

Use normalizing flow to learn conditional density $p_{Y|X}(\mathbf{y}|\mathbf{x})$. They use conditional prior $p_{Z|X}(\mathbf{z}|\mathbf{x})$ and a mapping $f_\phi : \mathcal{Y} \times \mathcal{X} \to \mathcal{Z}$, $\mathbf{z} = f_\phi(\mathbf{y}, \mathbf{x})$ bijective in $\mathcal{Y}$ and $\mathcal{Z}$. The likelihood model by the change of variable is

$$p_{Y|X}(\mathbf{y}|\mathbf{x}) = p_{Z|X}(\mathbf{z}|\mathbf{x}) \left| \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \right| = p_{Z|X}(\mathbf{z}|\mathbf{x}) \left| \frac{\partial f_\phi(\mathbf{y}, \mathbf{x})}{\partial \mathbf{y}} \right| \ , \tag{11.1}$$

where all the distribution are conditioned on $\mathbf{x}$.

They point out that by maximizing the likelihood through the $\mathcal{Z}$ space rather than directly the $\mathcal{Y}$ we do not bias the learning by our assumptions on the shape (family) of the $p_{Y|X}(\mathbf{y}|\mathbf{x})$ distribution to hand-craft the loss. So this can be seen as learning the loss.

They use all the tricks from RealNVP sec. 2 (coupling layers, sqeeze layers, split prior), Glow sec. 13.5 (invertible 1x1 convoluitions) and Flow++ sec. 13.6 (variational dequantization).

The conditioning is applied to the prior $p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu(\mathbf{x}), \sigma(\mathbf{x}))$, the split prior $p(\mathbf{z}_1|\mathbf{z}_0, \mathbf{x}) = \mathcal{N}(\mathbf{z}_1; \mu(\mathbf{z}_0, \mathbf{x}), \sigma(\mathbf{z}_0, \mathbf{x}))$, and in the coupling layers $\mathbf{y}_0 = s(\mathbf{z}_1, \mathbf{x})\mathbf{z}_0 + t(\mathbf{z}_1, \mathbf{x})$, $\mathbf{y}_1 = \mathbf{z}_1$.

They claim for binary data the variational dequantization proposed in Flow++ is not that great because the quantization prior is not bounded a propose to use something they call half-infinite noise. I'm not quite sure what they mean here but my feeling is they are getting away from the symmetrical Gaussian to something only positive and simple so they can simply recover the $\{0,1\}^D$ space.

The experiments are on single image super resolution (ImageNet32 and 64) and image-to-image translation and (DRIVE) meaning their conditioning variable $\mathbf{x}$ is always a full image so quite a strong signal. *Would work equally well if it were just a class indicator?*

12 Livne & Fleet: TzK: Flow-based Conditional Generative Model

Micha Livne and David Fleet. "TzK: Flow-Based Conditional Generative Model". In: (2019)

Notes taken: 3/7/2020

A bit awkward paper, I'm not sure I understood. They propose a compositional conditional generative model which incorporates task-specific conditioning. The principal idea is that they learn common features across multiple datasets and they then train a complimentary task-specific model over the common features to account for the particularities of the task (sub-domain) of interest. Intuitively, this should work better than training each task separately by providing more data from the other tasks an therefore learning better features *though I don't think they ever say this*. It can either be trained end-to-end if you know the tasks of interest or you first train the common model and then you train the specific model.
This intuition seems easy and rather obvious but the paper and architecture are much less clear.

They use flow architecture and map the target variable $\mathbf{t}$ to a latent variable $\mathbf{z}$ via a smooth invertible mapping $\mathbf{z} = f_\theta \mathbf{t}$. They further condition the latent state $\mathbf{z}$ on a latent code $\mathbf{k} = \{\mathbf{k}^i\}_{i=1}^K$, where $\mathbf{k}^i = (e^i, \mathbf{c}^i)$ with $e^i \in \{0, 1\}$ and $\mathbf{c}^i \in \mathbb{R}^C$. They refer to $\mathbf{k}^i$ is *knowledge type* i with e^i being the *existence* of the knowledge of the specific type i and $\mathbf{c}^i$ the latent code of the knowledge i. The knowledge classes can interact through the common latent representation $\mathbf{z}$.

They assume independence between the knowledge types and adopt an encoder decoder framework through which they can learn the many distributions they need. This is where I get lost because the architectures (and the reasons for making it so) seem very unclear to me. Particularly, check figure 1 and equations 3-7 in the paper which are the core of the thing. There are many distributions that need to be learned and I'm not sure which, when and why.

13 Short dirty notes for more papers

13.1 Rippel, Adams: High dimensional density estimation

Oren Rippel and Ryan Prescott Adams. "High-Dimensional Probability Estimation with Deep Density Models". In: *arXiv:1302.5125 [cs, stat]* (2013)

Notes taken: 22/2/2020

Good review of various density estimation methods organised in tow groups:

- graphical models, e.g. Boltzmann machine, directed belief networks, etc.
- manifold learning, e.g. Gaussian latent variable model, kernel PCA, auto-encoder neural network

Inference is often costly and not scalable to high dimensions. They propose the *deep density model* (DDM) using deep learning for bijective transformations of the observed space.

The structure they propose is quite different from modern flow architectures. They suggested to have an invertible *decoder* $f_\theta : \mathcal{Z} \subseteq \mathbb{R}^D \to \mathcal{X} \subseteq \mathbb{R}^D$ such that the transformation of the observed data $f_\theta^{-1} : \mathcal{X} \to \mathcal{Z}$ has a simple factorised distribution

$$p_Z(\mathbf{z}) = \prod_i^D p(z_i) \ . \tag{13.1}$$

The decoder f shall be a stacking of standard sigmoidal layers $\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$.

They also have non-bijective *encoder* $g_\phi : \mathcal{X} \to \mathcal{Z}$.

Their objective is composed of three terms:

- divergence term making sure that after the encoding transform g_ϕ the empirical distribution of the latent representation matches the target distribution of choice. They work with Beta distributions and do this by calculating analytically the KL divergence between the base Beta and the a Beta fit to the empirical $\mathbf{z}$'s. *this seems complicated*
- invertibility measure which makes sure that the learned $\mathbf{W}$'s are invertible. This is based on the condition number but as they say, in high dimensions virtually all matrices are orthogonal.
- Reconstruction loss checking that the encoder-decoder chain reconstructs well. This is based on entropy which I don't really understand why.

They have some interesting ideas for experiments:

- to test their density estimation they check the density assigned to train data and test data which should both be high and distorted data which should have low density
- they train on MNIST digit 9 and check the density for 6 (should be low) and rotated 6 (should be high)

They mention the possiblity to extend to supervised learning with calibrated classificaiton predictions.

13.2 Germain et al, Masked Autoencoder for Distribution Estimation (MADE)

Mathieu Germain et al. "MADE: Masked Autoencoder for Distribution Estimation". In: *arXiv:1502.03509 [cs, stat]* (2015)

Notes taken: 22/2/2020

Standard autoencoders over binary data optimizing cross-entropy per dimension are equivalent to maximizing the factorised log likelihood. However, the true distribution is unlikely to be factorised. In contrast, any distribution can be decomposed into the product of *autoregressive* conditionals where the distribution for each dimension is conditioned on the previous dimensions.

They propose to multiply (element-wise) the weight matrices in the encoder and decoder by binary masks that will block the paths so that the outputs only depend on previous dimensions of the inputs. The idea is fairly simple and the trick is only how to construct these automatically so that it actually really blocks the path as needed - they give simple heuristic that can be initialised randomly.

They further make this more complicated by randomly permuting the order of the dimensions during the training when the order has no ture meaning.

13.3 Oord et al., PixelRNN

Aäron van den Oord et al. "Pixel Recurrent Neural Networks". en. In: *Proceedings of the 33rd International Conference on MachineLearning*. 2016, p. 10

Notes taken: 22/2/2020

Model the distribution of an image by expressing the joint distribution as a product of conditional distributions where we condition the distribution of each pixel on the prevoius pixels (to the left and top).

To model the recurences they use RNN (with LSTM layers) and masked CNN where the filters are masked so that they don't peak to the future pixels.

The paper discusses plenty of architectures for the RNN or CNN models achieving this. But the point is that the images have to be generated sequentially by passing the outputs again and again through the networks to produce more and more pixels.

13.4 Oord et al., Conditinal generations with PixelCNNs

Aaron van den Oord et al. "Conditional Image Generation with PixelCNN Decoders". In: *arXiv:1606.05328 [cs]* (2016)

Notes taken: 22/2/2020

Extends the previous PixelCNN by conditioning all the autoregressive distributions on a common vector **h** giving some high-level description of the image (e.g. class). This can be either global, so used for conditioning all pixels or local and then would be applied only to some pixels through masking.

13.5 Kingma and Dhariwal: Glow

Diederik P. Kingma and Prafulla Dhariwal. "Glow: Generative Flow with Invertible 1x1 Convolutions". In: *arXiv:1807.03039 [cs, stat]* (2018)

Notes taken: 23/2/2020

Good motivation of generative modelling and major pros and cons of flows as compared to autoregressive, VAEs and GANs in the intro.

Very clear intro to flows (hinting to the fact that continuous variables are always discretized by digitalization process so they add uniform noise to the observation to make them continuous again).

Takes realNVP architecture (section 2) and replaces

- batch norm with *actnorm* - for large images the minibatch has size 1 and therefore batchnorm would be very unstable. Actnorm normalizes initial activations per channel by the mean and std calculated over initial minibatch and then treats these as learnable parameters
- dimension permutation with invertible 1x1 convolution - this is easy to invert and get Jacobian determinant
- affine coupling layer with additive layers - in the factor-out architecture they split the features along channels

Good experimental results. They claim the 1x1 convolution does better than random or reverse permutations reaching higher log likelihood values. Generations from CelebA look good. They also show interpolations in the latent representations and these look very good suggesting the latent space is very smooth. They also have a nice trick for semantic manipulation: they simply calculate average latent vectors for images with and without some discrete feature (e.g. smile) and then move along the difference of these two. Looks good.

They speak about somehow reducing the temperature of the model but I'm not sure what temperature they speak about. This seems like post-hoc reduction in the variance of the latent space for neater 4sampling.

13.6 Ho et al: Flow++

Jonathan Ho et al. "Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design". In: *arXiv:1902.00275 [cs, stat]* (2019)
Notes taken: 23/2/2020

Builds on Glow (secttion 13.5) and claims that transforming the digitized data to continuous by adding uniform noise (*dequantization*)

$$\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{u}, \qquad \mathbf{u} \sim \mathcal{U}(0,1)^D \tag{13.2}$$

and maximizing the likelihood $p(\tilde{\mathbf{x}})$ is not great. It does prevent the model from collapsing to the discrete data, however, it asks the model to assign uniform density to to unit hypercubes around the data $\mathbf{x} + [0,1)^D$ which is not natural for a smooth function approximator.

They therefore propose to learn the most reasonable distribution of the additive $\mathbf{u}$ through approximation $q(\mathbf{u}|\mathbf{x})$ which itself is modelled as a flow $\mathbf{u} = q_x(\boldsymbol{\epsilon})$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Doing this they don't optimize the likelihood but a variational bound but they claim that even the uniform deqauntization boils down to optimizing a variational bound where we approximate the true $p(\mathbf{u}|\mathbf{x})$ by the uniform distribution.

They add some more tweaks to the architecture. They design more complex affine layers with extra invertible nonlinearity babsed on Logistic CDF of the data which requires learning plenty more parameters. They bring attention and gating to the networks acting in the coupling layers. They claim these help but are less important than learning of the $\mathbf{u}$ distribution.

13.7 Magdon-Ismail, Atiya: Density Estimation Using Multilayer Networks

M. Magdon-Ismail and A. Atiya. "Density Estimation and Random Variate Generation Using Multilayer Networks". en. In: *IEEE Transactions on Neural Networks* 13.3 (2002), pp. 497–520

Notes taken: 23/2/2020

Rather old paper. They propose to learn the data distribution via NN as non-parametric class. They rely on the CDF rather than the pdf as pdf is just a derivative of the CDF which should be able to obtain from the neural net.

In the univariate case $x \in \mathbb{R}$ the r.v. $y = CDF(x)$ is uniform on $(0, 1)$. They propose to learn the nn as the CDF, that is the outputs of the network should be distributed uniformly. They pose it as a supervised learning problem and to each x they associate an $y \sim Uni(0,1)$ and optimize the network. They generate new ys after every forward-backward pass to allow it to learn a truly uniform distribution not just fit the outputs.

In the multivariate case $\mathbf{x} \in \mathbb{R}^d$ the variable $\mathbf{y} = CDF(\mathbf{x})$ is not necessarily uniform.
(*I had though it is still marginally uniform but not independent*). They construct targets for the learning as the fraction of the training data smaller than each of the training points. The density is simply the derivatve of the learned CDF.

13.8 Uria, Murray, Larochell: RNADE

Benigno Uria et al. "RNADE: The Real-Valued Neural Autoregressive Density-Estimator". In: *arXiv:1306.0186 [cs, stat]* (2014)

Notes taken: 23/2/2020

Good review of traditional approaches to density estimation in the intro.

Extends previous Neural Autoregressive Distribution Estimator (NADE) which modelled discrete distribution to the continuous case.

The model is autoregressive relying on the generally valid decomposition of probability density by the product rule (chain rule)

$$p(\mathbf{x}) = p(x_1) \prod_{d=2}^{D} p(x_d|\mathbf{x}_{<d}) \tag{13.3}$$

In NADE each conditional is given by a feed-forward neural net with one hidden layer $\mathbf{h}_d \in \mathbb{R}^H$

$$p(x_d|\mathbf{x}_{<d}) = sigm(\mathbf{v}_d^T \mathbf{h}_d + b_d), \qquad \mathbf{h}_d = sigm(\mathbf{W}_{<d}\mathbf{x}_{<d} + \mathbf{c}) \ . \tag{13.4}$$

The parameters are tied across the dimensions with $\mathbf{W}_{<d}$ being the first $d-1$ columns of a shared weight matrix $\mathbf{W}$.

For the Real-valued neural autoregressive distribution estimator (RNADE) the conditionals are modelled as mixture of Gaussians so that the outputs of the network are the mixing fractions (softmax), component means and component standard deviations ($\log \sigma$). Otherwise the architecture pretty much follows NADE including the parameter tying. The architecture is a rather shallow network, probably due to the date of drafting and comp resources available at the time.

In addition to mixture of Gaussian they also tried mixture of Laplacians, claiming it may be more appropriate for some types of data. They only experimented with relatively low diemansional data and compared to simple baselines such as mixture of Gaussians optimised by EM and it performed reasonably well.

13.9 Papamakarios, Pavlakou, Murray: MAF

George Papamakarios et al. "Masked Autoregressive Flow for Density Estimation". In: *arXiv:1705.07057 [cs, stat]* (2018)

Notes taken: 23/2/2020

View autoregressive model as a normalizing flow and use MADE to avoid the sequential looping. Can stack multiple autoregressive flows on top of each permuting the order of the dimensions which breaks the arbitrary ordering creating troubles in autoregressive models.

After stating the main idea it shows how the autoregressive models can be interpreted as flows and how this links to inverse autoregressive flows of Kingma (section 8).

Experiments are not amazing so seem more honest. Also does conditional density estimation and generation by conditioning all the densities in the autoregression on an side variable y by $p(\mathbf{x}|\mathbf{y}) = p(x_1|\mathbf{y})\prod_{d=2}^{D} p(x_d|\mathbf{x}_{<d}, \mathbf{y})$

13.10 Wang, Wang: Neural Gaussian Copula for Variational Autoencoders

Prince Zizhuang Wang and William Yang Wang. "Neural Gaussian Copula for Variational Autoencoder". en. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, 2019, pp. 4332–4342

Notes taken: 23/2/2020

They argue that one of the problems in VAEs is the mean field assumption for the approximate posterior $q(\mathbf{z}|\mathbf{x})$. They propose to use copulas to capture the dependencies. Though the idea of using copulas in general may be interesting, in their case I don't see the beauty because it essentially boils down to learn $q(\mathbf{z}|\mathbf{x}) = N(\mu(\mathbf{x}), \Sigma(\mathbf{x}))$ with full covariance matrix where $\Sigma(\mathbf{x}) = L(\mathbf{x})^T L(\mathbf{x})$ are outputs of the NN.

13.11 Weise et al.: Copula and Marginal flows

Magnus Wiese et al. "Copula & Marginal Flows: Disentangling the Marginal from Its Joint". In: *arXiv:1907.03361 [cs, stat]* (2019)

Notes taken: 23/2/2020

I don't think I understand. They first that you distribution estimate should coincide with the true distribution in cumulative distribution function and that if you have some tail beliefs you should be able to factor them in.

Then they show that current flows do not necessarily learn the tail probabilities satisfactorily (I don't understand the proofs).

Then they suggest to model the density as the product of copula and marginal densities

$$p(x_1, x_2) = p(x_2|x_1)p(x_1) = c(F_1(x_1), F_2(x_2))p(x_1)p(x_2) \ , \tag{13.5}$$

where c is the copula density and F_i are the marginal cumulative distribution functions.

They then propose flows for the marginals and the copula. Somehow it seems to me that they assume you know the copula over the uniform vector $u_i = F_i(x_i)$ a priori and you just need to flow it through and combine with the data distribution.

They only develop and experiment with bivariate case but claim this can be extended to higher dimansions via vine copula.

13.12 Tagasovska et al.: Copulas as High-Dimensional Generative Models

Natasa Tagasovska et al. "Copulas as High-Dimensional Generative Models: Vine Copula Autoencoders". In: *arXiv:1906.05423 [cs, stat]* (2019)

Notes taken: 24/2/2020

Train normal neural autoencoder with lower-dimensional latent space and learn a copula distribution over the latent space (post-hoc) to be able to generate new data by decoding the z samples.

They use the same density representation as above in equation (13.5) (product of copula density and the marginals) generalizing to $d > 2$ variables. To estimate the marginals they use Gaussian kernel density estimation. They use parametric and non-parametric (claim this is better since more flexible) vine copulas based on pair-copula constructions (pair-wise dependencies conditioned on other variables) to estimate the copula density. The standard estimation procedure is sequential constructing *conditioned pseudo-observations* to be able to evaluate the conditional copulas. This somehow reminds me of autoregressive models but I'm not quite sure about the link. They use the R implemented rvinecopulib to do the estimation.

13.13 Dinh et al.: RAD

Laurent Dinh, Jascha Sohl-Dickstein, Razvan Pascanu, et al. "A RAD Approach to Deep Mixture Models". In: (2019)

Notes taken: 24/2/2020

Flow model where they intorduce an extra latent variale k which shall cater for multiple modes (clusters) in the data. The flow transformation f thus has to be no-longer invertible but only piecewise invertible over partitions of data associated to the invidual clusters. They consider the partitions to be disjoint (means hard-clustering / strict separation which seem quite strong and restrictive assumption).

Otherwise just extending RealNVPs (section 2) with the extra f_K clustering function. They not this is difficult to optimise due to discontinuities but they say they solve it by gating layers. Not sure how.

On experiments seems to perform better than standard RealNVP

13.14 Tabak and Turner: Family of Nonparametric Density Estimation Algos

E. G. Tabak and Cristina V. Turner. "A Family of Nonparametric Density Estimation Algorithms". en. In: *Communications on Pure and Applied Mathematics* 66.2 (2013), pp. 145–164

Notes taken: 24/2/2020

One of the original normalizing flow papers. Suggests to maximize the likelihood via the parametrized flow map $x = f_\beta(z)$ from a standard Gaussian latent $\mathbf{z}$. They also propose to stak the transformations to combine simple transforamtions into something more complex. However, they don't go around it by NN but rather something that looks like kernel density estimation.

13.15 Ardizzone: Guided image generation, conditional INN

Lynton Ardizzone, Carsten Lüth, et al. "Guided Image Generation with Conditional Invertible Neural Networks". In: *arXiv:1907.02392 [cs]* (2019)

Notes taken: 24/2/2020

Builds upon their paper on inverse problems (section 9). It proposes conditional generations where the condition enters directly as inputs to the $t()$ and $s()$ networks in the coupling layers (this is ok, since these are not iverted). This is easy for something like MNIST and the class one hot vectors. For something more complicated, the condition (e.g. image) may need to be pre-processed by a conditioning network, either some pre-trained network allowing to extract reasonable features or trained together directly with the cINN. For MNIST they train with c the one-hot vectors for the digit classes and can then conditionally generate from each digits tweaking the style by changing $\mathbf{z}$. They can do the reverse in fixing the style in $\mathbf{z}$ and change c to generate different digits or even style transfer by passing $\mathbf{x}$, getting the corresponding $\mathbf{z}$ and changing $\mathbf{c}$.

They also tweak the style of the single digits by moving in the $\mathbf{z}$ space along PCA latent vectors (learned post-hoc over $\mathbf{z}$s of training data).

They have another experiment on coloring images where the conditioning info is the grayscale image and the colorization comes from learned distribution. I don't quite understand how this works (what is *Lab color space*?) but the results seem rather impressive to me. Not so much to ICLR reviewers, it seems ;)

References

[1] Lynton Ardizzone, Jakob Kruse, et al. "Analyzing Inverse Problems with Invertible Neural Networks". In: *arXiv:1808.04730 [cs, stat]* (2019) (cit. on p. 36).

[2] Lynton Ardizzone, Carsten Lüth, et al. "Guided Image Generation with Conditional Invertible Neural Networks". In: *arXiv:1907.02392 [cs]* (2019) (cit. on p. 47).

[3] Laurent Dinh, David Krueger, and Yoshua Bengio. "NICE: Non-Linear Independent Components Estimation". In: *ICLR (Workshop)*. 2015 (cit. on pp. 19, 24).

[4] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density Estimation Using Real NVP". In: *ICLR*. 2017 (cit. on pp. 16, 19).

[5] Laurent Dinh, Jascha Sohl-Dickstein, Razvan Pascanu, et al. "A RAD Approach to Deep Mixture Models". In: (2019) (cit. on p. 46).

[6] Mathieu Germain et al. "MADE: Masked Autoencoder for Distribution Estimation". In: *arXiv:1502.03509 [cs, stat]* (2015) (cit. on pp. 34, 41).

[7] Jonathan Ho et al. "Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design". In: *arXiv:1902.00275 [cs, stat]* (2019) (cit. on p. 43).

[8] Emiel Hoogeboom et al. "Integer Discrete Flows and Lossless Compression". In: *Advances in Neural Information Processing Systems 32 (NIPS 2019)*. 2019 (cit. on p. 28).

[9] Diederik P. Kingma and Prafulla Dhariwal. "Glow: Generative Flow with Invertible 1x1 Convolutions". In: *arXiv:1807.03039 [cs, stat]* (2018) (cit. on p. 42).

[10] Durk P Kingma et al. "Improved Variational Inference with Inverse Autoregressive Flow". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. 2016 (cit. on p. 33).

[11] Manoj Kumar et al. "VideoFlow: A Conditional Flow-Based Model for Stochastic Video Generation". In: (2020) (cit. on p. 38).

[12] Micha Livne and David Fleet. "TzK: Flow-Based Conditional Generative Model". In: (2019) (cit. on p. 40).

[13] David J C MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2005 (cit. on p. 28).

[14] M. Magdon-Ismail and A. Atiya. "Density Estimation and Random Variate Generation Using Multilayer Networks". en. In: *IEEE Transactions on Neural Networks* 13.3 (2002), pp. 497–520 (cit. on p. 44).

[15] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel Recurrent Neural Networks". en. In: *Proceedings of the 33rd International Conference on MachineLearning*. 2016, p. 10 (cit. on pp. 34, 42).

[16] Aaron van den Oord et al. "Conditional Image Generation with PixelCNN Decoders". In: *arXiv:1606.05328 [cs]* (2016) (cit. on p. 42).

[17] George Papamakarios, Theo Pavlakou, and Iain Murray. "Masked Autoregressive Flow for Density Estimation". In: *arXiv:1705.07057 [cs, stat]* (2018) (cit. on p. 45).

[18] Danilo Jimenez Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". In: *arXiv:1505.05770 [cs, stat]* (2016) (cit. on p. 23).

[19] Danilo Jimenez Rezende, George Papamakarios, et al. "Normalizing Flows on Tori and Spheres". In: *arXiv:2002.02428 [cs, stat]* (2020) (cit. on p. 22).

[20] Oren Rippel and Ryan Prescott Adams. "High-Dimensional Probability Estimation with Deep Density Models". In: *arXiv:1302.5125 [cs, stat]* (2013) (cit. on p. 41).

[21] Kyle Siegrist. *Random: Probability, Mathematical Statistics, Stochastic Processes*. https://www.randomservices.or (cit. on p. 4).

[22] E. G. Tabak and Cristina V. Turner. "A Family of Nonparametric Density Estimation Algorithms". en. In: *Communications on Pure and Applied Mathematics* 66.2 (2013), pp. 145–164 (cit. on p. 46).

[23] Natasa Tagasovska, Damien Ackerer, and Thibault Vatter. "Copulas as High-Dimensional Generative Models: Vine Copula Autoencoders". In: *arXiv:1906.05423 [cs, stat]* (2019) (cit. on p. 46).

[24] Dustin Tran et al. "Discrete Flows: Invertible Generative Models of Discrete Data". In: *Advances in Neural Information Processing Systems 32 (NIPS 2019)*. 2019 (cit. on p. 25).

[25] Benigno Uria, Iain Murray, and Hugo Larochelle. "RNADE: The Real-Valued Neural Autoregressive Density-Estimator". In: *arXiv:1306.0186 [cs, stat]* (2014) (cit. on p. 44).

[26] Prince Zizhuang Wang and William Yang Wang. "Neural Gaussian Copula for Variational Autoencoder". en. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, 2019, pp. 4332–4342 (cit. on p. 45).

[27] Magnus Wiese, Robert Knobloch, and Ralf Korn. "Copula & Marginal Flows: Disentangling the Marginal from Its Joint". In: *arXiv:1907.03361 [cs, stat]* (2019) (cit. on p. 45).

[28] Christina Winkler et al. "Learning Likelihoods with Conditional Normalizing Flows". Version 1. In: (2019) (cit. on p. 39).

Index